

RL-TR-92-52, Vol I (of two)
Final Technical Report
April 1992



AD-A256 242



SOFTWARE RELIABILITY, MEASUREMENT, AND TESTING Software Reliability and Test Integration

Science Applications International Corp. (SAIC)
Research Triangle Institute (RTI)

James A. McCall and William Randell (SAIC)
Janet Dunham and Linda Lauterbach (RTI)



Reproduced From
Best Available Copy

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

92 10 14 045

398290
92-27095



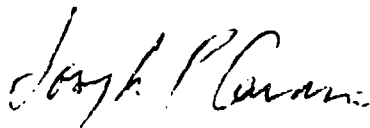
25px

Rome Laboratory
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

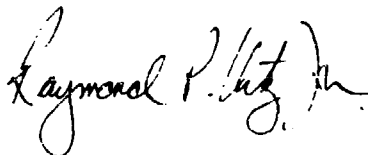
RL-TR-92-52, Vol I (of two) has been reviewed and is approved for publication.

APPROVED:



JOSEPH P. CAVANO
Project Engineer

FOR THE COMMANDER:



RAYMOND P. URTZ, JR.
Technical Director
Command, Control, & Communications Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL(C3CB), Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave Blank)		2. REPORT DATE April 1992		3. REPORT TYPE AND DATES COVERED Final Sep 86 - Dec 89	
4. TITLE AND SUBTITLE SOFTWARE RELIABILITY, MEASUREMENT, AND TESTING Software Reliability and Test Integration				5. FUNDING NUMBERS C - F30602-86-C-0269 PE - 62702F PR - 5581 TA - 20 WU - 63	
6. AUTHOR(S) James A. McCall and William Randell (SAIC) Janet Dunham and Linda Lauterbach (RTI)					
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Science Applications International Corp. (SAIC) 19260 Campus Point Drive, San Diego CA 92121 Research Triangle Institute (RTI) PO Box 12194, Research Triangle Park NC 27709				8. PERFORMING ORGANIZATION REPORT NUMBER N/A	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Rome Laboratory (C3CB) Griffiss AFB NY 13441-5700				10. SPONSORING/MONITORING AGENCY REPORT NUMBER RL-TR-92-52, Vol I (of two)	
11. SUPPLEMENTARY NOTES Rome Laboratory Project Engineer: Joseph P. Cavano/C3CB/(315) 330-4063					
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.				12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) This effort integrated software reliability, measurement, and test techniques in terms of prediction, estimation, and assessment. Experiments were conducted to compare six testing techniques and to measure the effect of software product and process variables on software reliability. A guidebook was produced to help program managers control and manage software reliability and testing. Error/anomaly and code reviews were the test techniques found to be the most effective at the unit level; branch testing and code reviews were the most effective at the CSC level. NOTE: Rome Laboratory/RL (formerly Rome Air Development Center/RADC)					
14. SUBJECT TERMS Software Reliability, Software Measurement, Software Testing				15. NUMBER OF PAGES 246	
				16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL		

TABLE OF CONTENTS

<u>Chapter</u>	<u>Page</u>
1.0 INTRODUCTION	1
1.1 Purpose	1
1.2 Scope	1
1.3 Objectives of Project	2
1.3.1 Software Measurement and Reliability Prediction	2
1.3.2 Software Testing and Reliability Estimation	3
1.3.3 Data Analyses and Recommendations	3
1.3.4 Software Reliability and Testing Guidebook	3
1.4 Approach	3
1.4.1 Test Techniques, Tools and Projects	4
1.4.2 Experiment Goals and Design	4
1.4.3 Reliability Prediction Data Collection	10
1.4.4 Software Testing and Evaluation	12
1.4.5 Test Data Collection	13
1.5 Executive Summary	15
1.5.1 Software Reliability and Testing Data	16
1.5.2 Software Reliability Prediction Experiment	16
1.5.3 Software Testing Experiment	16
1.5.4 Integrated Guidebook for Software Reliability and Testing	17
1.5.5 Integrated Reliability Management System "IRMS"	17
1.6 Organization of Report	17
2.0 SURVEYS	20
2.1 Software Projects Survey	20
2.1.1 Candidate Projects for Consideration	20
2.1.2 Evaluation Criteria	21
2.1.3 Selected Software Projects	21
2.1.4 Software Project Materials	25
2.1.5 Deleted	
2.1.6 Lessons Learned	28
2.2 Testing Techniques Survey	29
2.2.1 Candidate Techniques for Consideration	29
2.2.2 Evaluation Criteria	30
2.2.3 Selected Testing Techniques	34
2.3 Test & Support Tools Survey	36
2.3.1 Candidate Tools for Consideration	37
2.3.2 Evaluation Criteria	38
2.3.3 Selected Test & Support Tools	38
3.0 SOFTWARE RELIABILITY PREDICTION	44
3.1 Technical Approach	44
3.2 Data Collection Resources	46
3.2.1 Computer Systems	46
3.2.2 Support Software	46

TABLE OF CONTENTS (CONTINUED)

<u>Chapter</u>	<u>Page</u>
3.2.2.1 Automated Measurement System	48
3.2.2.2 4th Dimension DBMS	48
3.2.3 Personnel	48
3.2.4 Instruction Manual	51
3.2.5 Project Documentation	51
3.3 Reliability Prediction Methodology	51
3.3.1 Reliability Models	53
3.3.2 Reliability Computations	53
3.3.2.1 Application RPFOM	58
3.3.2.2 Development Environment RPFOM	58
3.3.2.3 Requirements and Design RPFOM	58
3.3.2.4 Implementation RPFOM	59
3.3.3 Software Reliability Prediction	60
3.4 Data Collection Framework	62
3.4.1 General Procedures	62
3.4.2 Life-Cycle Phase Worksheets	63
3.4.3 Database Development	63
3.5 Refinements to the SRPEG	65
3.6 Data Collection Results	69
3.6.1 Effort Summary	69
3.6.2 RPFOM Numbers	70A
3.7 Findings and Conclusions	70A
3.7.1 Support Tools and Database	75
3.7.2 Software Projects and Materials	75
3.7.3 Data Collection and Calculation	95
4.0 SOFTWARE TESTING EXPERIMENT	99
4.1 Technical Approach	99
4.2 Test Resources	102
4.2.1 Computing System	102
4.2.2 System Software	103
4.2.2.1 Communications	103
4.2.2.2 Operations Systems	103
4.2.2.3 Compiler	103
4.2.2.4 Data Management and Analysis	103
4.2.3 Test/Support Tools	103
4.2.3.1 DEC Test Manager	103
4.2.3.2 SDDL	103
4.2.3.3 RXVP-80	106
4.2.3.4 CPU Use Procedures	106
4.2.4 Software Code Samples	106
4.2.5 Personnel	108
4.2.6 Tester Instructions	108
4.3 Testing Techniques	110
4.3.1 Dynamic Techniques	112
4.3.1.1 Branch Testing	113

TABLE OF CONTENTS (CONTINUED)

<u>Chapter</u>	<u>Page</u>
4.3.1.2 Functional Testing	114
4.3.1.3 Random Testing	115
4.3.2 Static Techniques	115
4.3.2.1 Code Review	116
4.3.2.2 Error & Anomaly Detection	116
4.3.2.3 Structure Analysis	117
4.4 Reliability Estimation	118
4.4.1 Reliability Estimation Number (Model 2)	118
4.4.2 REN for Test Environments	120
4.4.2.1 Average Failure Rate During Testing (FT1)	120
4.4.2.2 Failure Rate at End of Testing (FT2)	120
4.4.2.3 Test Effort (TE)	120
4.4.2.4 Test Methodology (TM)	120
4.4.2.5 Test Coverage (TC)	121
4.5 Experiment Design and Conduct	121
4.5.1 Experiments	123
4.5.1.1 Unit Test Level	124
4.5.1.2 CSC Integration Test Level	124
4.5.2 Empirical Studies	125
4.5.3 Pilot Experiment	125
4.5.4 Communications and Monitoring	128
4.5.5 Applying the Techniques	129
4.5.5.1 Preparing and Executing the Tests	129
4.5.5.1.1 Test Preparation	129
4.5.5.1.2 Test Execution	133
4.5.5.1.3 Stopping Rules	133
4.5.5.2 Test Analysis	133
4.5.5.3 Data Collection and Organization	138
4.5.5.3.1 Data Collection	138
4.5.5.3.2 Data Organization	138
4.6 REN Results	142
4.6.1 Average Test Failure Rate (FT1)	144
4.6.2 Test Effort (TE)	144
4.6.3 Test Methodology (TM)	144
4.6.4 Test Coverage (TC)	146
4.6.5 REN Calculations	147
4.7 Tester Profiles	147
5.0 EXPERIMENT FINDINGS	150
5.1 Scope of Analyses	150
5.2 Analyses of the Test Data	150
5.2.1 Descriptive Analyses	150
5.2.1.1 Sample Description	150
5.2.1.2 Unit Testing	151
5.2.1.2.1 Single Test Technique Description	151

TABLE OF CONTENTS (CONTINUED)

<u>Chapter</u>		<u>Page</u>
	5.2.1.2.2 Test Technique Pair Description	162
	5.2.1.3 CSC Testing	166
	5.2.1.3.1 Single Test Technique Description	166
	5.2.1.3.2 Test Technique Pair Description	173
	5.2.1.4 Tester Profiles	177
5.2.2	ANOVA Results	177
	5.2.2.1 Process	184
	5.2.2.2 Single Technique Effectiveness Results	186
	5.2.2.3 Single Technique Coverage Results	187
	5.2.2.4 Single Technique Effort Results	188
	5.2.2.5 Single Technique Efficiency Results	190
5.2.3	General Observations	191
5.2.4	SAS Outputs: GLM Results	192
5.2.5	SAS Output: CHI-SQUARE Results	192
5.2.6	Error Summary Tables	192
5.3	RPFOM Exploratory Analysis	192
5.3.1	Data	193
	5.3.1.1 Error Data	193
	5.3.1.2 Metrics Data	193
	5.3.1.3 Missing Data	196
5.3.2	Analytical Approach	198
5.3.3	Results	198
	5.3.3.1 Modularity	198
	5.3.3.2 Complexity	201
5.3.4	Summary and Conclusions	201
APPENDIX C Unit-level Analysis Data Files		216
APPENDIX D CSC-level Analysis Data File		229

Accession For	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification _____	
By _____	
Distribution/ _____	
Availability Codes	
Dist.	Avail and/or Special
A-1	

EVALUATION

The concepts needed to understand reliability are not fully developed. Many of the important issues are too broad for a single-focused treatment and must be explored from different angles. This is especially true for understanding software reliability because software is intangible and difficult to understand in its own right—adding reliability only complicates the subject.

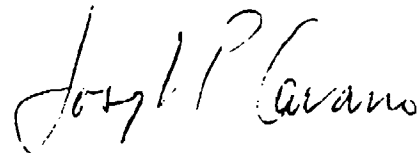
The goal of this report is to help bridge the gap between what management can control and what production needs to do. Although the book addresses software reliability and testing, the approach could be generalized to producing quality in other domains. More precisely, this report lays the foundation for the bridge to software reliability—it attempts to show how to determine factors that affect reliability across the life cycle and how to quantitatively evaluate the process of developing high reliability software so that one can improve upon the process in the future. The prediction and estimation numbers produced for software reliability are more valuable for comparing with other projects and in tracking progress toward continual quality improvement than in their absolute values. Further research in experimentally applying reliability measures across software development, review and test processes is necessary to validate the numbers. If you are not interested in improving quality on a long-term basis, then this report may not be especially helpful.

Although the measures described in this report produce exact software reliability numbers for fault density or failure rate, these numbers must be used with discretion because the proper relationship with specific levels of reliability have not yet been proven. There is no magical formula for deriving reliability predictions or assessments because software engineering does not yet have the necessary theory upon which to develop such equations. This does not mean that empirical observations cannot be used to develop a discipline. Instead, the reader is cautioned that the empirical observations made during this effort were not extensive enough to prove their validity over all projects. For example, the measures related to the development environment try to relate various characteristics of an environment to their individual impact on reliability. In the projects studied, there was not enough diversity in these characteristics to enable exact predictions at that level. Although data analyses lead to equations, the results are not appropriate across the complete range of possible outcomes—in fact, low values for the 'Dc' metric produce erroneous negative numbers. It is better to treat this metric at a more global level (i.e., organic, semi-detached or embedded) as shown in Metric Worksheet 1A.

In facing such situations, choices had to be made between the theoretical or ideal state and providing suggestions on how a typical organization could

customize, develop and use reliability measures, tailored to their unique procedures. This report leans to the practical side of measurement by showing the role that reliability prediction and estimation could play in the future. Purists might be disappointed in this.

The path to higher reliability and better testing is not always easy. To improve a process, change is required. If you don't intend to change your current procedures, this report may not be of much value. On the other hand, if you plan to experimentally apply the techniques described in this report, consistently observe results over several projects, and tailor the techniques and measures for your organization, then, hopefully, you will see an improvement in your software's reliability.

A handwritten signature in cursive script, reading "Joseph P. Cavano". The signature is written in dark ink and is positioned above the printed name.

Joseph P. Cavano

1.0 INTRODUCTION

1.1 Purpose

The purpose of this report is to describe the results of a research and development effort to integrate and improve the application of software reliability measurement and testing techniques. This is the final report of the project. This effort was performed under Contract Number F30602-86-C-0269 for the U.S. Air Force Rome Air Development Center (RADC).

1.2 Scope

Science Applications International Corporation (SAIC) and Research Triangle Institute (RTI) performed a formal software testing experiment utilizing code samples from two previously developed AF/DoD software systems as the experimental vehicles. These code samples were tested at the unit and Computer Software Component (CSC) integration test levels in accordance with DoD 2167. Each code sample was tested employing six different testing techniques which are representative of the current state of the practice. A common set of test/support tools were utilized. All testing was performed in accordance with an experimental framework which included formal experiments and empirical studies refined from an initial pilot. The test results were measured and analyzed in order to make recommendations about the error detection capability (effectiveness) of the testing techniques, the test coverage achieved, and the test effort expended. An additional part of the experiment was to compute the estimated reliability (REN, Reliability Estimation Number) of the final software product.

An independent software reliability study was conducted to measure characteristics of the software testing code samples and the complete software systems, and to improve the software reliability data collection procedures and forms. The collected metrics were used to compute the predicted reliability (RPFOM, Reliability Prediction Figure of Merit) of the final software product. Exploratory analyses were performed on these RPFOMs with resulting refinements to the metric multipliers, based on the experiment and study results.

Testing strategies have been developed from the analyses of the test results, and limitations and weaknesses are identified in the testing techniques which were utilized. Potentially useful new testing techniques are described. Finally, recommendations are made for future experimentation in software reliability testing.

Inputs to this effort were the RADC Software Test Handbook (STH) [2] and the RADC Software Reliability Prediction and Estimation Guidebook (SRPEG) [3]. The STH provides guidance on selecting appropriate state-of-the-art testing techniques to achieve a chosen testing confidence level. The SRPEG provides a methodology to predict software reliability based on software system characteristics, and analyzes test results to provide an estimate of future reliability for the final product.

The final output of this effort is the RADC Software Reliability Measurement and Test Guidebook, which is Volume 2 of this Final Report. Physically, the new guidebook contains the entire SRPEG, in a revised format with appropriate metric and procedural refinements, plus selected and appropriately revised sections of the STH.

Tools, techniques, raw data, analyses and software system documentation are being delivered to RADC where they will form the basis of a repository for future experimentation.

1.3 Objectives of Project

The objective of this research and development project was to integrate software reliability measurement and testing techniques to provide improved techniques for both software testing and reliability measurement in terms of prediction and estimation. This includes the design and conduct of a software testing experiment and empirical study for comparing test techniques and for measuring the effects of software process and product variables on software reliability. The results of this research also provide additional data for the refinement of the RPFOM and the REN.

Results of these experiments and studies provide a quantitative basis for recommendations to Air Force acquisition managers concerning improved methods for:

- a. Choosing test techniques.
- b. Allocating test effort among test levels and/or portions of software.
- c. Determining cost trade-offs in testing.
- d. Predicting and estimating software reliability.

These refinements and recommendations are incorporated into an integrated and improved software reliability and testing guidebook for the acquisition managers. This guidebook provides:

- a. Instructions for collecting metric data on software systems and analyzing the data to predict and estimate reliability.
- b. Guidance on selecting appropriate state-of-the-practice testing techniques.

1.3.1 Software Measurement and Reliability Prediction

The objectives of this activity were to perform and evaluate the software reliability prediction methodology contained in the SRPEG, and to obtain quantitative data that can be meaningfully analyzed and interpreted in order to:

- a. Contribute to the RADC software reliability database.
- b. Refine the reliability prediction model.
- c. Integrate the software reliability prediction methodology with techniques for software testing.

The primary elements of this study were to collect, compute and analyze software measurement data on selected software test projects in accordance with the software reliability prediction methodology as documented in the SRPEG. The data to be collected includes measures of Application Type, Development Environment, Anomaly Management, Traceability, Quality Review Results, Language Type, Modularity, Complexity and Standards Review Results. Computations include the RPFOM Numbers at the System and Computer Software Configuration Item (CSCI) levels during applicable Software Development Life Cycle phases for entire test projects. Computations also are performed for individual test samples. Analyses include exploratory regressions on the computed RPFOM numbers in order to refine the equations in the reliability model contained in the SRPEG.

1.3.2 Software Testing and Reliability Estimation

The objectives of this activity were to apply and evaluate commonly used software testing techniques, as specified in the STH, in accordance with the Software Reliability and Test Integration Plan (SRTIP) [6] and the Software Test Plan (STP) [7]. The SRTIP documents the design and conduct of the experiments and empirical studies for comparing test techniques and for measuring the effects of software process and product variables on software reliability. The STP describes the test program required to implement the SRTIP.

To accomplish these objectives, code samples were selected from two different software development projects and then tested using six different test techniques. The test techniques studied were random testing, functional testing, branch testing, code review, error and anomaly detection, and structure analysis. In addition, quantitative measures of software test effectiveness and efficiency were obtained during the testing process. Measures were obtained for two levels of testing: unit and CSC.

1.3.3 Data Analyses and Recommendations

Data analyses were conducted on the reliability data and on the test measurements and results. These activities involved the graphical presentation of data from simple descriptive statistics, analyses of variance (ANCOVA), empirical analyses of the test techniques, and from comparisons of the software reliability numbers with test results.

The analyses were oriented towards providing answers to specific questions of interest which were formulated from primary goals of the study. Test technique effectiveness, test effort and test coverage were addressed both within a test level and across test levels, for single techniques and for test strategies. Tester variability was also addressed. Exploratory analyses of the reliability data were conducted with the goal of making refinements to the software reliability prediction models.

Recommendations for testing strategies were made from the results of these analyses with the goal of writing an integrated guidebook for the software reliability measurement and testing.

1.3.4 Software Reliability and Testing Guidebook

Refinements to the present RADC methodologies for software reliability and testing, and recommendations resulting from the experiments and empirical studies were incorporated into an RADC Software Reliability and Testing Guidebook for Air Force acquisition managers. This new guidebook represents the integration and updating of the STH and SRPEG. The revised and integrated guidebook provides guidance on selecting state-of-the-practice testing techniques, and provides instructions for collecting metric data on software development projects and analyzing the data to predict and estimate the future reliability of the final product.

1.4 Approach

The Statement of Work (SOW) [1] outlines the approach to be taken in achieving the study goals and allocates the work to seven tasks (see Figure 1.1): selecting software projects, tools, and test techniques; designing the software testing experiments and

empirical studies; collecting raw data for computing the RPFOM on software development projects and code samples; gathering raw test data and measurements during software testing; computing the REN and reducing and analyzing the collected test data; interpreting the statistical results and making recommendations; and documenting findings in the updated and integrated guidebook. The principal activities for each of these tasks are shown in Figure 1.2.

1.4.1 Test Techniques, Tools and Projects

Our first task was to identify a candidate set of test techniques and tools to be used to support the software measurement and software testing to be conducted for the experiments. In order to best ensure the effectiveness of this selection, analysis was conducted at the same time concerning the software systems to be used.

The detailed analysis for this task began with a survey of existing software environments. Specifically, the survey collected data on:

- a. Testing techniques used to achieve reliable software.
- b. Tools used for software testing.
- c. Existing Air Force and DoD software development projects to which these tools and techniques may be applied.

We surveyed nine representative testing techniques as candidates for the experiments, based on their usage in industry, their applicability to the modern programming environment, and their ability to aid in the testing of computer systems. These techniques perform either static analysis or dynamic analysis of the software. They are described in Section 2.2.

Twenty-two candidate testing tools were surveyed for their applicability, power, and availability. The tools we have identified are classified either by the testing techniques they automate or as general test support tools. They are described in Section 2.3.

Twelve software projects were identified as candidates for the experiments, based on size, complexity, and type of application. They are a representative sample of Air Force systems in general development today. Each project is described in Section 2.1.

1.4.2 Experiment Goals and Design

The design adopted for this study was specifically geared toward providing results useful to acquisition managers of Air Force software. To ensure a sound, comprehensive design, the initial design was enhanced with inputs and reviews by experts in experiment design with human subjects and statistics and enhanced with inputs and review by industry and academic experts in the fields of computer science, software testing, experiment design with human subjects, statistics, and software metrics. [17]

A standardized approach to the experimental design was taken, based on earlier work by Basili and Reiter [18]. In following this approach, a precise definition of experimental goals in the form of specific questions to be answered was developed from the objectives declared in the SOW and contents of the initial guidebooks. These questions were organized into "questions of interest."

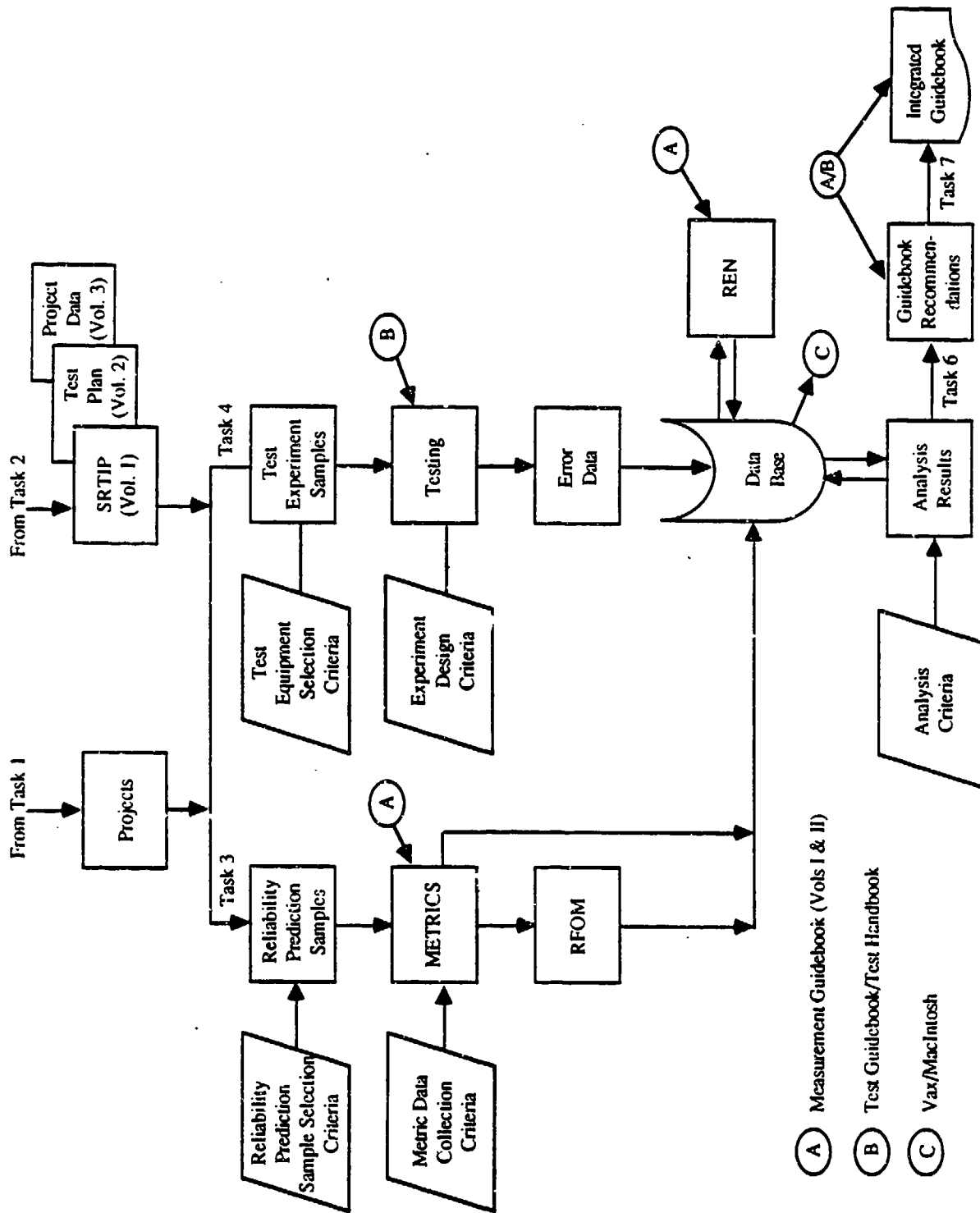


Figure 1.1 SRM TIT Task Flow Diagram

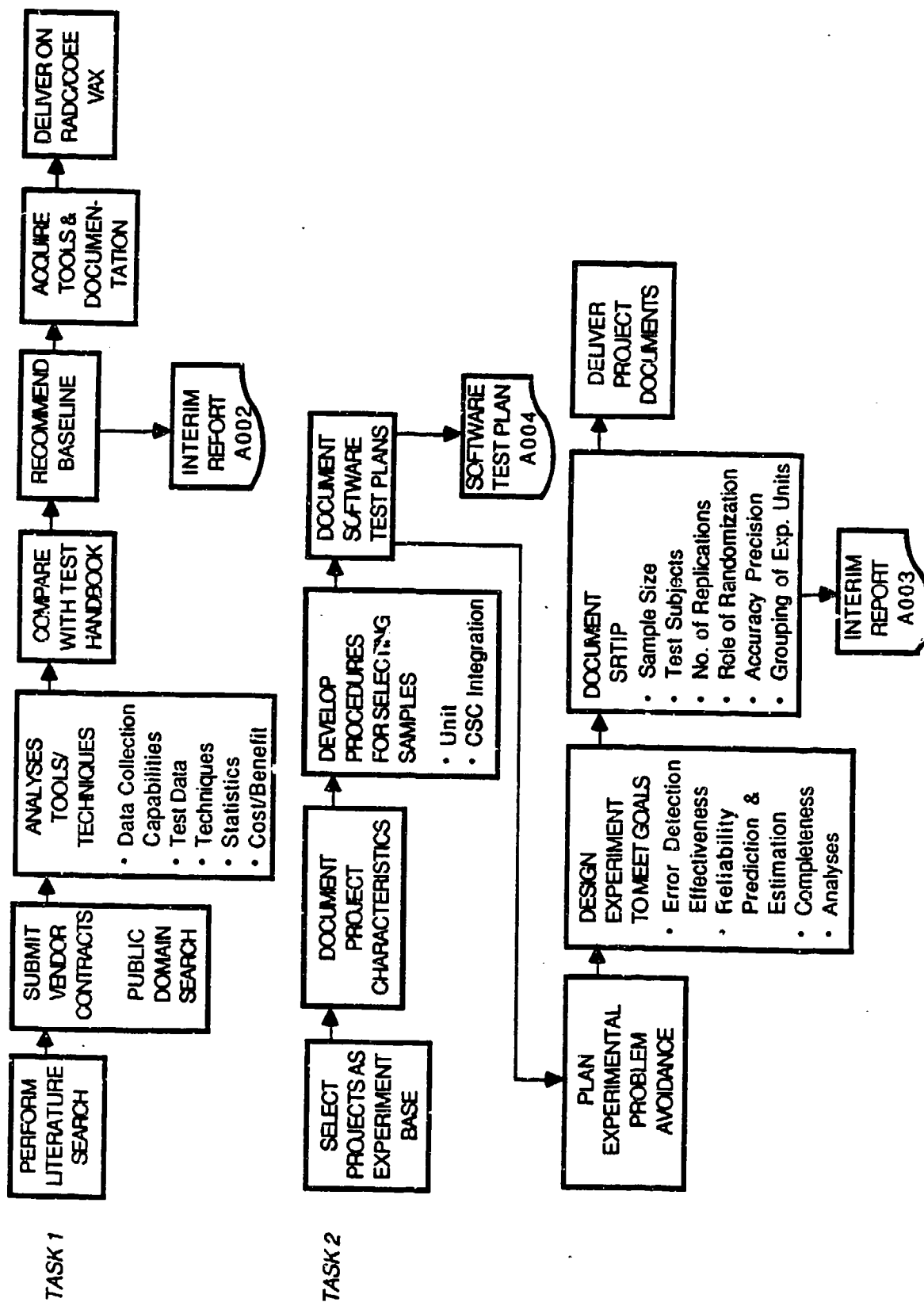
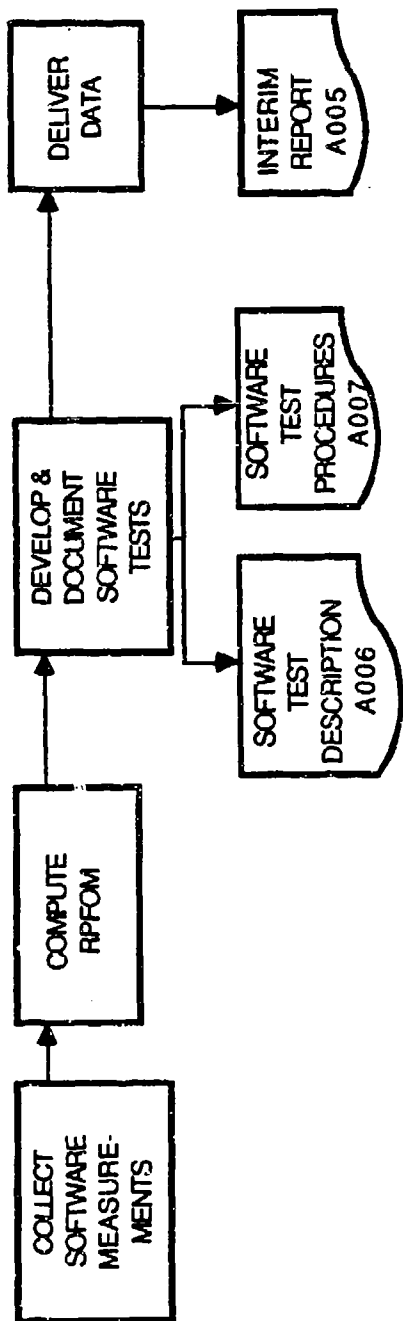


Figure 1.2. SRMTIT Activity Chart

TASK 3



TASK 4

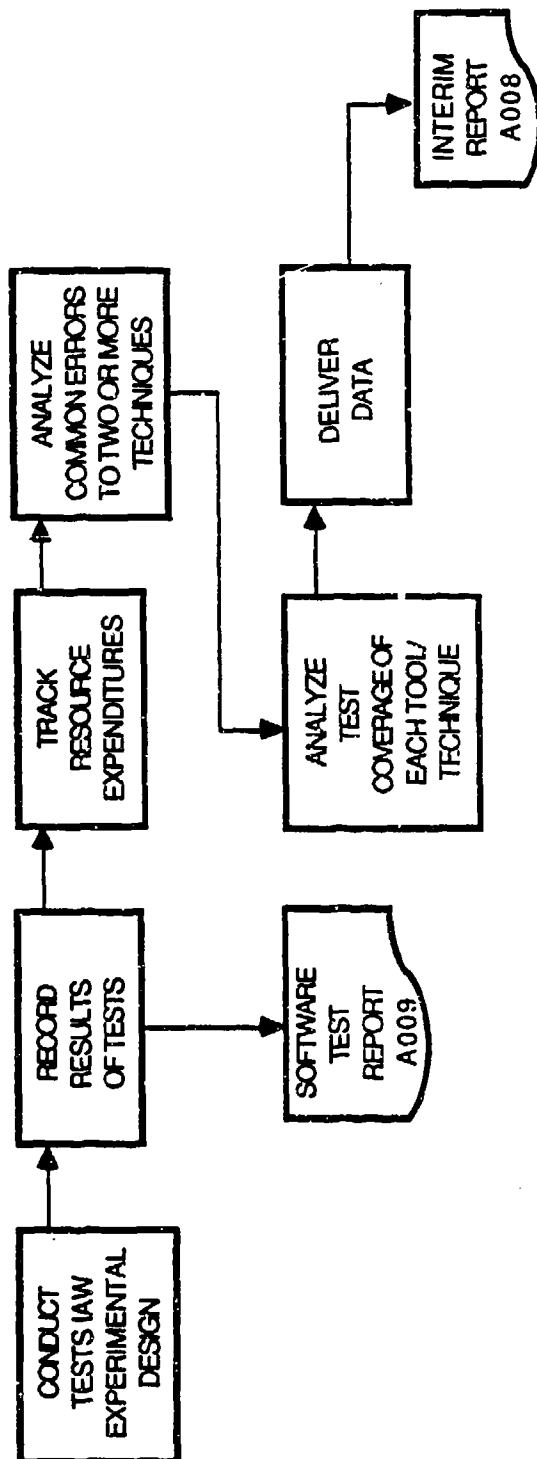


Figure 1.2. SRMTIT Activity Chart (Continued)

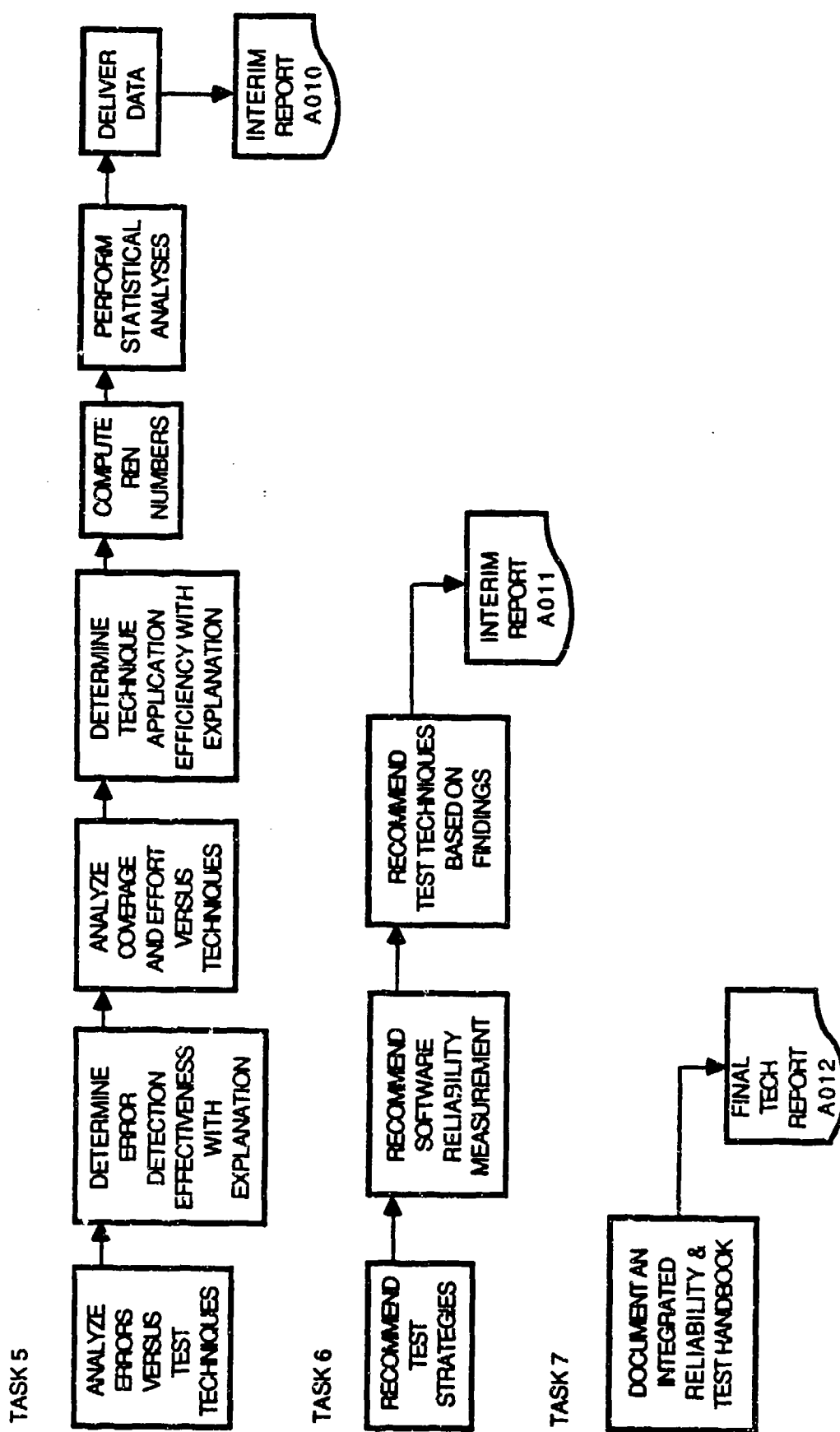


Figure 1.2. SRMTIT Activity Chart (Continued)

Candidate statistical designs were evaluated in terms of how well they would test the chosen questions of interest given the available projects, test techniques, tools, and other resources. Experiments to address all aspects covered in the two input guidebooks are beyond the scope and resources of this study. Thus, the design was tailored to address as much of the information in these two documents as possible, while controlling the experimental variables to the extent necessary to preserve the statistical soundness of the design.

A combination of experiments and empirical studies were selected to best meet the objectives of this study. These studies include application of the selected testing techniques and measurement of their relative effectiveness and efficiency. They are summarized here and detailed in Chapter 4.

Test techniques were separated into two categories: deterministic techniques and nondeterministic techniques. Deterministic techniques are those for which one can determine without doubt that applying a given test technique to a given code sample will find a given error. (Note that while one can determine error without doubt, the potential for human error in the determination process still exists. Thus deterministic does not imply fool-proof.) Nondeterministic techniques are those in which variability across such factors as test personnel and software characteristics can have profound effects on the effectiveness of the technique.

This distinction between test techniques was made to conserve time and resources. Techniques which can be highly and consistently automated and are less dependent upon tester expertise were seen as deterministic and suitable for empirical study. Nondeterministic techniques require experiments with several testers employing the test technique to the best of their ability on a given code sample. Experimental results will show whether applying the technique usually an/or consistently finds a given error.

The experiment addresses the following:

- a. A study of three dynamic test techniques: functional, random, and branch testing. Data analyses are by percent known errors observed at each test level, and by percent known errors at each test level. (Absolute number of errors are recorded for descriptive purposes. However, since each sample most likely will not have the same total number of errors, meaningful comparisons across samples can be made only on a percentage basis.)
- b. A study of one static test technique: code review. Data analyses are by percent of known errors detected and by percent of known errors detected at a test level.
- c. The measurement of each technique's relative efficiency, in terms of the number of discrepancy reports filed, on a time reference basis.

The empirical study addresses the following:

- a. A study of two static test techniques: structure analysis and error and anomaly detection. Data analyses are by percent of known errors detected and by percent of known errors detected at a test level.

- b. The establishment of the predictive validity of the RPFOM for the code samples tested by test technique.
- c. The measurement of each technique's relative efficiency, in terms of the number of discrepancy reports filed (errors located), on a time reference basis.

These studies were performed initially as a small-scale pilot experiment. The overall experiment design and methodology was verified in the pilot. This is done to increase the validity of the subsequent full-scale experiment and attendant findings and recommendations.

Since the empirical studies address deterministic procedures, only two testers are needed to apply each test technique chosen for the empirical study. The experiments address nondeterministic procedures and employ four testers. Samples consist of code from the actual product development.

The experiments are designed as Latin Squares and involve four testers repetitively testing code samples with different test techniques. The variability in results due to the nondeterministic nature of the techniques can be averaged over all the testers. This increases confidence that what is measured is a test technique's performance, not a tester's performance.

The questions of interest to be addressed by the experiments and the empirical studies span the static and dynamic testing techniques, the software reliability measures, and the experimental and empirical studies. These questions are categorized by goals and are easily converted to formal statistical hypotheses. General categories of test technique evaluation questions to be answered include measures or dependent variables to be used in the evaluation of test effectiveness, test effort, test coverage, fault or error source, and error severity within a test level. Evaluation across test levels also is performed. Fault or error impact provides a measure of the extent to which the software must be modified in order to correct the fault or error.

Data is collected to provide insight into the effectiveness and efficiency of different test techniques at different test levels, per the test objectives: i.e., the test effort required, test coverage goals, and software error characteristics.

1.4.3 Reliability Prediction Data Collection

The framework for software reliability prediction and estimation is shown in Figure 1.3. Software reliability prediction is a static process which makes a quantitative statement about future reliability as a function of metrics. Software reliability estimation makes a quantitative observation of attained reliability using dynamic test data and extrapolates it to the operational environment. This part of the study is concerned with reliability prediction. Reliability estimation is discussed in Section 1.4.5.

This study represented the first independent application of the SRPEG and therefore provided the basis for correcting deficiencies in the SRPEG methodology. The SRPEG methodology for software reliability prediction data collection was simplified in order to meet the study objectives. This involved condensing and integrating the tasks, procedures, and worksheets of the SRPEG into a set of easy-to-follow detailed instructions for data collectors that correlate precisely with the sequence of availability of data sources that are customarily produced during the software

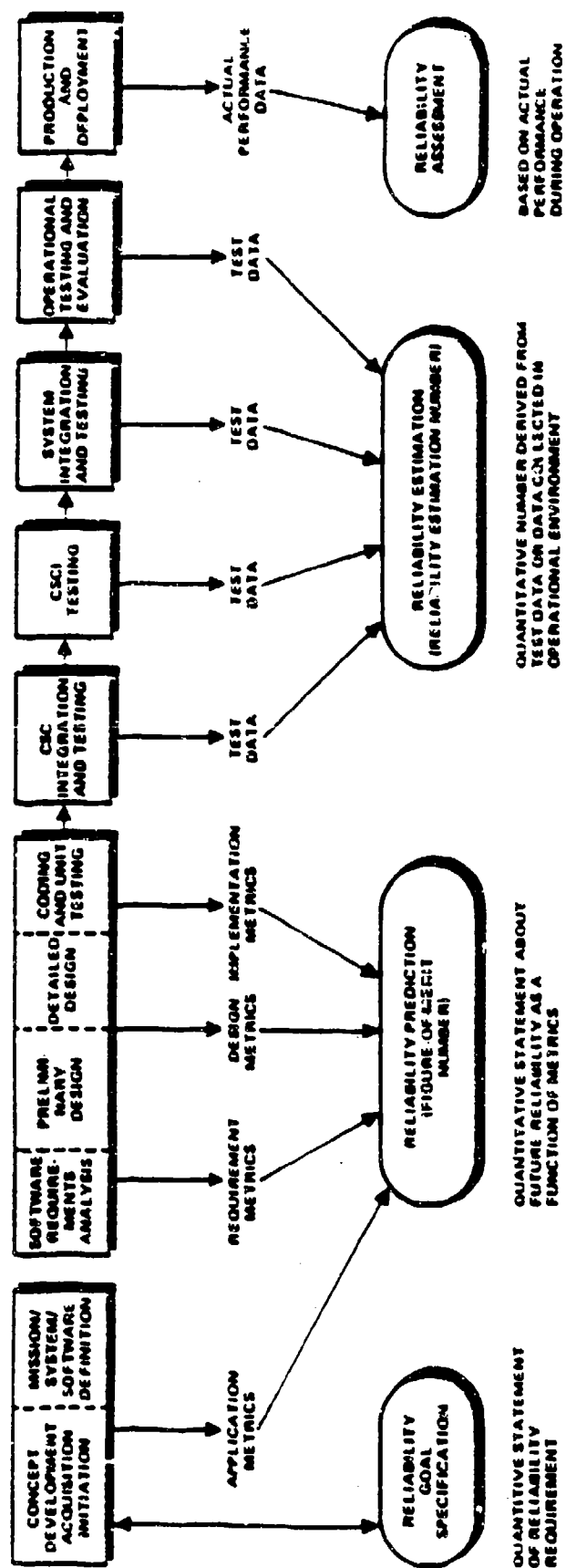


Figure 1.3. Framework for Software Reliability

development life cycle. A number of corrections, clarifications, and refinements to the SRPEG were accomplished during this process. These changes provide a foundation for revisions to the SRPEG which are incorporated in the integrated RADC Software Reliability and Testing Guidebook.

The focus of software reliability data collection is a set of four equations, each defining a RPFOM at a different stage of the software development life cycle. These are discussed in detail in Chapter 3. Unlike the SRPEG, which provides techniques for reliability prediction from system definition through operational testing and evaluation, the simplified instructions are applicable only through the coding and unit testing phase of the life cycle, as addressed by the experiments. They include step-by-step instructions, worksheets and answer sheets which support RPFOM data collection and computation. They were applied to the test projects in a manner which emulates a real-world application. Consequently, only project data sources were utilized which would have logically existed at the software life-cycle phase corresponding to the metrics of interest. This approach was necessary in order to meaningfully test the utility of the reliability prediction methodology.

The RPFOM data to be collected for each test project is specified in metric worksheets. Each worksheet targets a specific metric, software life-cycle phase, and software component level. The data collected was manually recorded on metric answer sheets prior to entry into an automated database in order to facilitate data entry and thus reduce the impact of multiple users accessing a single workstation. Each answer sheet supports all worksheets corresponding to a specific software component level and life-cycle phase.

One Application-type RPFOM was calculated for each test project. This baseline RPFOM, determined prior to initiation of software development, is an average fault density based on the principle application of the test project. Then the Development Environment RPFOM, which is a refinement of the baseline RPFOM, was calculated. It incorporates information pertaining to the software development environment of a system.

The Requirements and Design RPFOM was computed next. This is a refinement of the Development Environment prediction and incorporates information on software characteristics provided by system requirements and design documentation. The last RPFOM to be computed was the Implementation RPFOM. It represents a final refinement to the reliability prediction at the CSCI level and incorporates information on software characteristics derived from source code during coding and unit testing. Data collection for this RPFOM involved utilization of the RADC Automated Measurement System (AMS) on the Digital Equipment Corporation (DEC) VAX computer system for collection of many of the unit-level metric elements.

1.4.4 Software Testing and Evaluation

Software testing quantifies the correctness and completeness of the final product and its conformity to requirement. It also provides an environment in which software reliability estimation may occur. See Figure 1.2. Software testing was performed in accordance with the experiment goals and design described in Section 1.4.2.

Special tester instructions were prepared to ensure uniform application of the test experiment framework. Testers applied each testing technique in the order specified by the Latin Squares, utilizing the procedures contained in these instructions. Test project consultants assisted in pre-test preparation and post-test evaluation activities.

Testers created test cases, test procedures and test drivers from specifications provided by the consultants. A test harness was setup utilizing the DEC Test Manager (DTM) to provide inputs to the test driver and to capture and compare test outputs from the code samples under test. Chapter 4 contains a detail description of the software testing and evaluation process.

Preparing for test execution for each test sample included development of a test driver followed by test data preparation and formulation of expected results for each test technique. Test data preparation formulates test cases and the data to be input to the code sample. Test case preparation was not applicable to the static testing techniques. For the dynamic testing techniques, test preparation was supported by DTM and RXVP-80, a program code analyzer from General Research Corporation (GRC). The Software Design and Documentation Language (SDDL) tool was utilized to document the code sample source. The final step before test execution was to tailor the test driver, if needed, to suit any particular needs of the test cases for a given code sample and testing technique.

For the dynamic techniques test execution involved executing each code sample with prepared test cases and then collecting the results. For the static techniques, test execution constituted the execution of RXVP-80, as appropriate, on each code sample and the evaluation of the applicable hardcopy output according to the procedures of the given technique.

Test evaluation was performed by the testers for each dynamic testing technique to capture and report test effort and execution details (e.g., branch execution counts) and to determine the thoroughness of the testing (i.e., test coverage). For the static testing techniques, evaluation is an integral part of their execution. The concluding step was for each tester to evaluate the static and dynamic techniques to determine the unique errors found, both individually and by more than one technique, and whether each error found was a known development error or newly detected during the experiment.

1.4.5 Test Data Collection

Careful thought was given to the test technique and Reliability Estimation (REN) data to be collected and to the data collection procedures. General forms and procedures that support the test technique data collection activities were prepared for use by testers during test development and execution. They derive from requirements in DoD-STD-2167A for Test Description, Test Procedure and Test Report; from the SRPEG metric data collection forms for the REN; and from statistical data analysis requirements in the SRTIP related to discrepancy reporting, execution time, failure rate, test effort, test coverage and test methodology.

These collected test results and measurements were entered into a test database utilizing 4th Dimension (4D) from Acius, Inc. From there a subset of the collected data was converted to data analysis input files using StatView 512+ from Abacus Concepts, Inc. These files support the REN calculations and descriptive and statistical analyses. Their logical organizations were designed to meet the following objectives:

- a. To parallel the distinction between experiment activities (i.e., reliability measurement data collection vs. test technique and REN evaluation).
- b. To permit easy file update associated with these activities by multiple personnel at multiple sites.

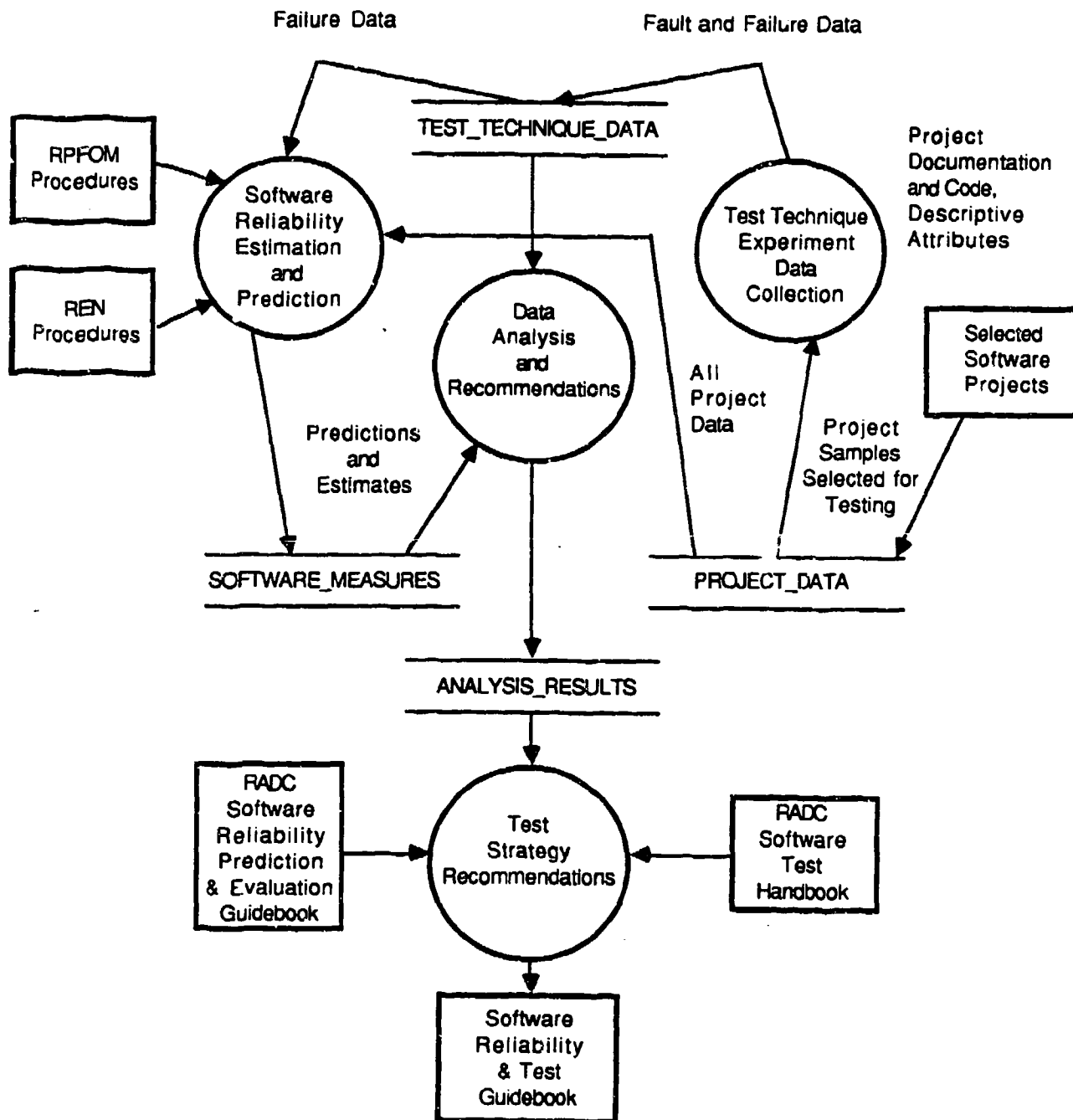


Figure 1.4. Relationship Between the Experiment and Resulting Guidebook

c. Refine the software reliability models for RPFOM.

One goal of these descriptive analyses is to characterize the observations of unique discrepancy reports detected as a percentage of known original project discrepancy reports plus those discovered during testing for each test technique at each test level it is applied. In addition, test effort and test coverage are evaluated both for test techniques applied singly and in strategies, with in test levels and across test levels. Tester variability is also addressed. These descriptive analyses were conducted using data collected during the pilot, the empirical study and the formal experiment. Specific questions that are answered are derived from the SRTIP. Descriptive analyses were performed for both unit and CSC integration levels.

Descriptive analyses were conducted to provide insight into the characteristics of the test data collected. The response variables of interest were then used with the ANOVA models dictated by the Latin Square designs chosen for the experiment. The ANOVA was conducted on the combined pilot and unit data, but not on the single CSC latin square. Identified data transformations were performed on the unit data before the analysis to ensure the best possible results. ANOVA tests were conducted on the effectiveness, effort and coverage for the test techniques. Information from those ANOVA results guides which further analyses are meaningful. These data are analyzed in a manner that provides answers to the questions of interest for which an ANOVA is specified. The same analyses were conducted for the formal experiment and the empirical study.

The goal of the exploratory analysis of the RPFOM equations is to investigate and improve the metric multiplier coefficients in the applicable software reliability model. Multiple regression analysis and partial correlations were used. These analyses take into account the metric data collected from the test projects during the experiments.

The Statistical Analysis System (SAS) was used for all ANOVAs due to the flexibility of the tool. StatView was used for descriptive and exploratory analyses for which it is very well suited.

Descriptive analyses include preparing histograms and plots and other data presentations for use in interpreting results into recommendations for testing strategies and contribute directly to the updating of the guidebook.

Statistical analysis of the test techniques guides the formulation of effective test strategies. Exploratory analysis of the software reliability measurement numbers provides the basis for refinements to the software reliability models for the RPFOM. Conclusions from these investigations and resulting recommendations are documented in Chapters 3 and 6. Chapter 6 also describes the application of these results and recommendations to the existing SRPEG and STH in order to produce the integrated Software Reliability and Testing Guidebook for Air Force acquisition managers.

1.5 Executive Summary

The important results of this effort can be summarized into five areas. Each area is briefly highlighted here with reference to the sections of the report where details can be found.

1.5.1 Software Reliability and Testing Data

Documentation and source code were collected for four software development projects as experimental vehicles for software reliability and testing experiments. Two of these software systems provided the basis for experimentally controlled observations about software reliability and testing as documented in the RADC Software Reliability Prediction and Estimation Guidebook [3] and the RADC Software Testing Handbook [2]. These observations were directed toward the integration of software and testing into a simple framework for use by Air Force acquisition managers and software developers. The software system database supported the conduct of the software testing experiments and the development of an integrated guidebook for making reliability predictions and estimations, and for selecting and applying state-of-the-practice testing techniques. Summary Software project data are presented in Chapter 2 of this report.

1.5.2 Software Reliability Prediction Experiment

The SRPEG includes a comprehensive guide to the application of the RADC software reliability prediction technology. It was recognized at the beginning of the current task that this document also included considerable supplemental background and application strategy in support of the technology. A detailed study of the RPFOM tasks, procedures, computations and worksheets contained there further showed this information to be far in excess of what would be needed in our study production environment.

It was decided that these components of the document should be reorganized and condensed into a straightforward set of step-by-step instructions for data collectors. Several discrepancies were found and corrected while evaluating this material. The resultant instruction manual guided the data collectors through each worksheet and answer sheet and presents the applicable equations (some of which are automated) in a simple sequence of computations. Details of this work are discussed in Chapter 3.

These new instructions were followed and even improved upon by our data collectors. Their application has shown them to be direct and easy to use. The only further refinements being needed were alternate procedures for special cases, subdivision of the procedures and forms by both metric and Life Cycle phase and an index table to locate and correlate them with the metric computations.

Data collection included the effort required to collect data and use the methodology. This is an important aspect of assessing the costs/benefits of the methodology.

All of these refinements are evaluated in Chapter 3 and are included in the new guidebook. Chapter 5 includes a study of the computed metric multiplier coefficients using the experiment data. The results of this study are included in the new guidebook. Still, a more comprehensive item-by-item evaluation of the SRPEG methodology is needed, but was outside the scope of the present study. The new guidebook should be a living document and incorporate all future advancement of the present software reliability prediction technology.

1.5.3 Software Testing Experiment

A plan and design was developed for the acquisition and analysis of software reliability measurement and test data within the structure of a formal experimental framework. Chapter 4 documents the design and conduct of these experiments and

empirical studies for comparing test techniques and for measuring the effects of software process and product variables on software reliability. The experimental design for the experiment was a significant element of the overall study. The approach taken, the techniques and tools selected, the cooperative team effort expended, the test instructions developed, the data requirements identified, the automated environment and cross-country network established all contribute to an excellent model for further experimentation. The assessment of each test technique from an input, process, output viewpoint, the instructions for how each tester would approach testing individual units and CSC's and the establishment of criteria (stopping rules) for test completion offer valuable information for testers and have been included in the test guidance portion of the guidebook (Vol II).

Results of these experiments and empirical studies are presented in Chapter 5. They provide a quantitative basis for recommendations to Air Force acquisition managers with regard to choosing test technique(s), allocating test effort among test levels and/or portions of software, and determining cost trade-offs in testing for Air Force software projects. These recommendations are incorporated into a guidebook for the acquisition managers.

The software test project data described above (Section 1.5.1) and the present experimental design may be reapplied to such studies, providing additional benefit to the Government. The scope of such studies can be increased by the utilization of data from additional software systems.

1.5.4 Integrated Guidebook for Software Reliability and Testing

A guidebook (Volume II of this report) was produced to allow software reliability engineers and software test engineers to practice the techniques developed during this research effort. Utilizing the data collected and findings derived from analysis, procedures are provided which allow reliability predictions and estimations to be made at various milestones during a software development project; additional procedures guide the selection and application of efficient and effective testing techniques.

Figure 1.5 illustrates how this new Guidebook is an integration and updating of two existing guidebooks: RADC Software Test Handbook [2] and the RADC Software Reliability Prediction and Estimation Guidebook [3]. The Software Test Handbook provides guidance on selecting appropriate state-of-the-practice testing techniques to achieve a chosen testing confidence level. The Software Reliability Prediction and Estimation Guidebook provides instructions for collecting metric data on projects and analyzing the data to predict and estimate the future reliability of the final product.

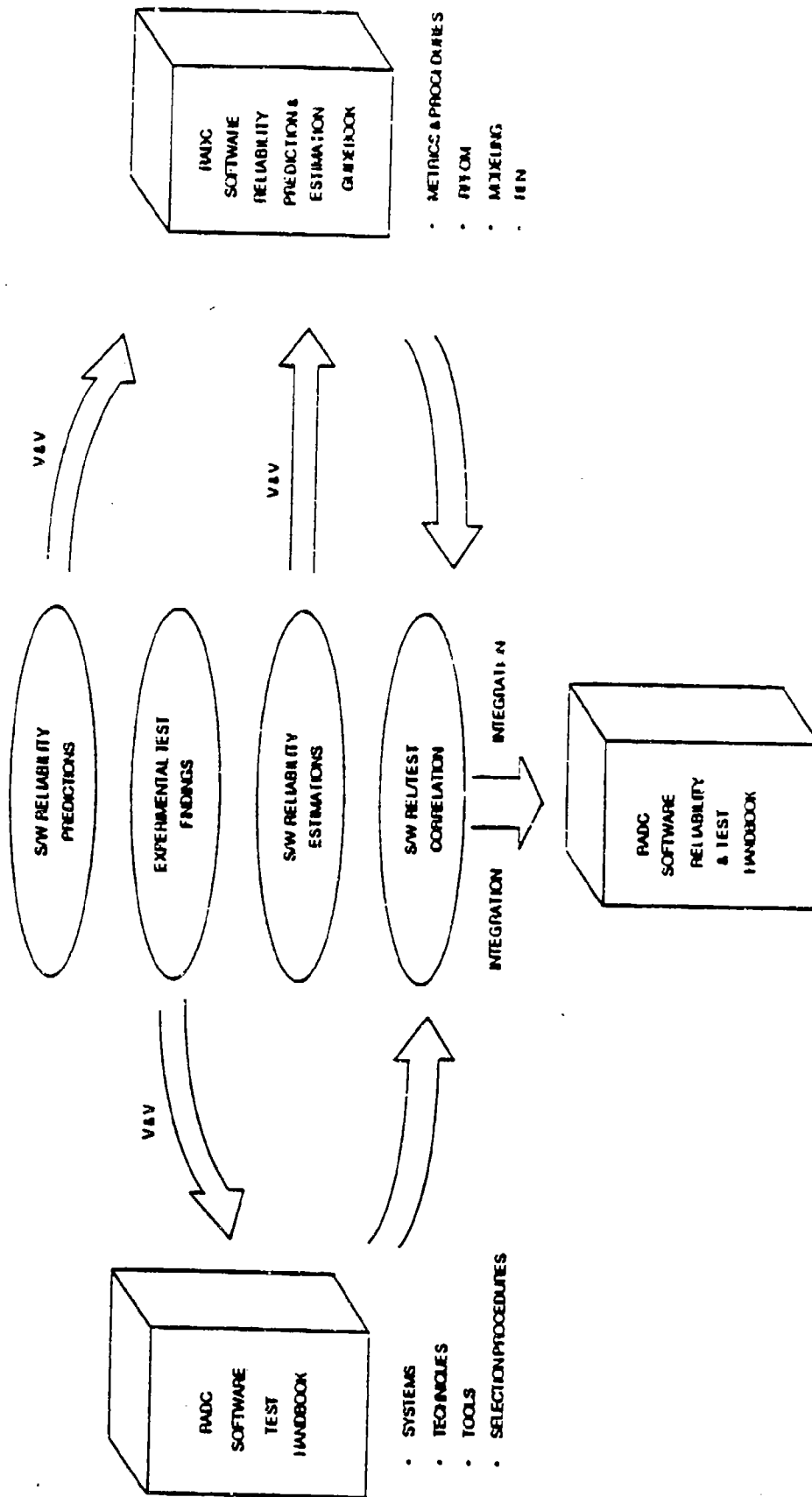
1.5.5 Integrated Reliability Management System (IRMS)

Enhancements to the IRMS were made during the contract effort and the IRMS was used as the central data management, analysis, and reporting component of the study. The IRMS is functionally illustrated in Figure 1.6.

1.6 Organization of Report

This document is Volume 1 of the Final Report. It is intended for those interested in the objectives, process and results of the Software Reliability Measurement and Test Integration Techniques Study (SRMTIT). It presents a structured view of the research and development process employed by SAIC and RTI in the conduct of this

Figure 1.5
INPUT TO THE GUIDEBOOKS



effort the form of the investigations made and all findings, conclusions and recommendations. The product of this study effort is a Software Reliability Measurement and Testing Guidebook which is intended for use by Air Force acquisition managers when establishing guidance in the reliability and testing of software systems. This guidebook is separately bound as Volume 2 of the Final Report.

Chapter 1 introduces the study effort and describes the study purpose, scope, objectives, and technical approach. It also provides an executive summary of the important results of this effort.

Chapter 2 describes the three surveys which were performed in order to select the software projects, testing techniques and testing tools which were utilized in the study. Selected projects, techniques and tools are identified here, along with rationale for their selection. All surveyed and rejected projects, techniques and tools are also identified.

Chapter 3 describes and discusses the software reliability prediction data collection activity and the calculation of the RPFOM. It describes the software reliability prediction metrics and detailed RPFOM computations at four software project levels: application, development environment, requirements and design, and implementation. It also describes the refinement of the data collection computations, procedures and forms as they were used in the experiment and are incorporated into the final guidebook. The resources which were utilized in this data collection process are identified here. In addition, this chapter contains the computed RPFOMs for the four software project levels and each of the test codes samples. It concludes with an evaluation of the study resources, RPFOM procedures and computations, technical activities and results. An effort table is provided to estimate and schedule Government applications of this technology.

Chapter 4 presents the framework of the software testing experiments and empirical studies. It discusses the conduct of the software testing procedures and the software reliability and testing methodologies which underly the experiment design. Each of the testing techniques are described there, along with test data and measures to be collected for test technique evaluation and for computation of the REN for each test sample. This chapter also identifies all of the resources utilized during the experiment. It concludes with a summary of the test data collection and evaluation by the testers and contains the computed RENs for each test sample.

Chapter 5 presents the form of the experiment data analyses, describes the findings, and discusses conclusions which can be made. All descriptive analyses and Analyses of Variance of the test technique data are described here. Also included in this chapter are exploratory analyses of the RPFOM for the test code samples.

Appendix A is a glossary of acronyms used in this report.

Appendix B is a bibliography of documents which are referenced or provide guidelines for this report.

2.0 SURVEYS

Information contained in this chapter represents the results of three technical surveys of available software development projects, testing techniques and tools for use in the experiments which were performed during the present study.

2.1 Software Projects Survey

Forty-three software development projects were surveyed. Of these, twelve software projects became candidates for the experiments, based on size, complexity, and type of application. They are a representative sample of Air Force and Department of Defense (DoD) systems in general development today.

2.1.1 Candidate Projects for Consideration

The candidate projects, in ascending order by size, are:

- a. Advanced Field Artillery Tactical Data System (AFATDS) Simulation/Stimulation (Sim/Stim) software. (10-50K) - Sim/Stim is designed to drive/test the AFATDS System Model under Test (SMT) by simulating nodes in the Brigade slice that are not present in the SMT and stimulating equipment within the SMT.
- b. Facility Automated Maintenance Management Engineering System (FAMMES) (10-50K) - A warehouse inventory control program.
- c. Mission Effectiveness Model (MEM) (10-50K) - MEM models the performance of the space segments of Strategic Defense Initiative (SDI) ballistic missile defense concepts.
- d. On-Board Electronic Warfare Simulator (OBEWS) (10-50K) - The OBEWS designed for Armament Division and Tactical Air Command is a dynamic simulation of a realistic Electronic Warfare (EW) threat environment designed to provide real time simulated threats to existing aircraft EW systems displays.
- e. Radiological Release Information System (RRIS) (10-50K) - A very reliable nuclear power plant safety system.
- f. SCENE (10-50K) - SCENE is a space defense scenario generator that models a variety of sensors and foreign launch missions under the Surveillance Command & Control Design Analysis and Engineering (SCCDA&E) contract.
- g. Boost Phase Stereo Processor (BPSP) (50-100K) - The BPSP is a realtime sensor data fusion system under the SCCDA&E contract. Its purpose is to accept observations of a booster in powered flight from multiple sensor sources and fuse these observations into an estimate of the current state (position and velocity of the booster, as well as projecting that state through to booster burnout.
- h. SIMSTAR Preprocessor (50-100K) - This program is a database preprocessing system for a large nuclear weapon effects simulator system.

- i. Automatic Test Control System (ATCS) (>100K) - The ATCS USAF project is a key component of the Aeropropulsion systems Test Facility used to test air breathing engines at the Arnold Engineering Development Center.
- j. Mideastern Command, Control and Communications (C3) and Communications Protection Plan (MC3 & CPN) (>100K) - The MC3 and CPN is a program with the US Navy to design, develop and implement a C3 and communications protection program for Saudi Arabia.
- k. National Training Center (NTC) (100K) - A large, real-time, multi-processor, range monitoring and control system.
- l. Architecture Design and Assessment System (ADAS) (<70K) - A set of engineering tools used for the simulation and study of electronic systems. This system includes two tools: Adasim and Csim. Each tool contains approximately 5K source lines. These two tools are functionally equivalent, aiding in the performance of Random Testing. In addition, the Adasim program is coded in Ada, which permits collection of information that will impact testing strategies to be used in Ada developments.

2.1.2 Evaluation Criteria

In order to be potential candidates the following basic criteria had to be met by each project surveyed:

- a. The projects must be unclassified USAF or DoD systems, or similar in mission characteristics.
- b. Source code must be available at different "snapshots" in time during the standard development process. This could include completed projects of which there were historical copies kept or on-going projects whose source code could be released.
- c. Documentation must be available, including requirement specifications, design documents and software test documents, or equivalent.
- d. Development/target hardware must be accessible.

Table 2.1 gives an overview of the candidate projects in terms of these and additional criteria that were considered for project candidacy.

2.1.3 Selected Software Projects

Four of these candidate projects were selected, two for use in the experiment and two others available as backup. Table 2.2 categorizes these four projects by software categories contained in the STH.

Advanced Field Artillery Tactical Data System (AFATDS) Simulator/Stimulator (Sim/Stim) - AFATDS simulator/stimulator is a DoD simulation and stimulation software system that is designed to drive and test the system model under test (SMT) by simulating nodes in the Brigade slice that are not present in the SMT and stimulating equipment within the SMT.

SELECTED CRITERIA CANDIDATE PROJECTS	TYPE OF CONTRACT			SIZE OF PROGRAM				LANGUAGE USED				TYPE OF PROJECT																
	USAF	DoD	Other	<10K	10-50K	50-100K	>100K	FORTRAN	ADA	C	OTHER	EVENT CONTROL	PROCESS CONTROL	PROCEDURE CONTROL	NAVIGATION	FLIGHT DYNAMICS	ORBITAL DYNAMICS	MESSAGE PROCESSING	DIAGNOSTIC SOFTWARE	SENSOR/SIGNAL PROCESSING	SIMULATION	DATA BASE MANAGEMENT	DATA ACQUISITION	DATA PRESENTATION	DECISION/PLANNING AID	PATTERN/IMAGE PROCESSING	SYSTEM SOFTWARE	DEVELOPMENT TOOLS
ADAS (Architectural Design & Assessment System)	Y			Y					Y	Y			Y	Y				Y	Y	Y	Y					Y		Y
AFATDS (Advanced Field Artillery Tactical Data System)		Y			Y			Y					Y			Y			Y	Y	Y	Y					Y	
ATCS (Automated Test Control System)	Y						Y	Y				Y		Y		Y			Y	Y	Y	Y		Y		Y		Y
BFSP (Boost Phase Stereo Processor)	Y					Y		Y				Y		Y		Y			Y	Y	Y			Y			Y	
PANMUS (Facilities Automated Maintenance Mgmt. Engineering System)	Y				Y			Y				Y										Y		Y				
Midcourse Command, Control & Communications & Comm. Protection Plan		Y					Y	Y										Y				Y		Y		Y	Y	Y
MEM (Mission Effectiveness Model)	Y				Y			Y									Y	Y	Y	Y	Y			Y	Y			
NTC (National Training Center)		Y				Y			Y			Y							Y	Y	Y	Y	Y	Y	Y		Y	Y
OBIEWS (On Board Electronic Warfare Simulator)	Y				Y			Y					Y								Y	Y						
NRIS (Radiological Release Information System)			Y		Y			Y	Y									Y			Y	Y		Y	Y			
SCENB (Scenario Generator)	Y				Y			Y									Y		Y					Y	Y			
SIMSTAR Preprocessor	Y					Y		Y									Y	Y		Y	Y	Y			Y			

TABLE 2.1 CANDIDATE PROJECTS AND CHARACTERISTICS - PART 1 OF 2

SELECTED CRITERIA	DEVELOPMENT		TARGET		SOURCE CODE AVAILABLE								DOCUMENTATION AVAILABLE								STAGE IN LIFE CYCLE		SW MEASUREMENT DATA COLLECTED
	HARDWARE	OPERATING SYSTEM	HARDWARE	OPERATING SYSTEM	BEFORE UNIT TEST	AFTER UNIT TEST	BEFORE CSC TEST	AFTER CSC TEST	BEFORE CSC TEST	AFTER CSC TEST	BEFORE SYSTEM TEST	AFTER SYSTEM TEST	KB STD FOLLOWED	SOFTWARE REQ. SPEC	TOP LEVEL DESIGN SPEC	DETAILED DESIGN SPEC	SOFTWARE TEST PLAN	SOFTWARE TEST PROCEDURE	SOFTWARE TEST REPORT	SOFTWARE DEVELOPMENT PLAN	IN PROCESS	COMPLETED	
CANDIDATE PROJECTS	ADAS (Architectural Design & Assessment System)	Quadri PH 8050	UNOS	VMS	11/750 VAX	VMS					Y				Y	Y	Y	Y	Y	Y	Y	Y	
	APATDS (Advanced Field Auxiliary Tactical Data System)	750 VAX 4.2	VMS	VMS 4.2	750 VAX 4.2	VMS	Y	Y	Y	Y		Y		2167	Y	Y	Y	Y	Y	Y	Y	Y	Y
	ATCS (Automated Test Control System)	3287 Quadri	2.1	MPX	Quadri MPX	MPX						Y		1679	Y	Y	Y	Y	Y	Y	Y	Y	Y
	BPSP (Boost Phase Stereo Processor)	Micro Vax	VMS	VMS	Micro Vax	VMS					Y	Y		2167	Y	Y	Y	Y	Y	Y	Y	Y	Y
	PAMBERS (High Level Automated Mission Mgmt. Engineering System)	Micro Vax	VMS	VMS	Micro Vax	VMS					Y	Y			Y	Y	Y	Y	Y	Y	Y	Y	Y
	Midwestern Command, Control & Communications & Comm. Protection Plan	VAX 760	VMS	VMS	VAX 760	VMS	Y	Y	Y	Y	Y	Y		1679	Y	Y	Y	Y	Y	Y	Y	Y	Y
	MEM (Mission Effectiveness Model)	VAX 760	VMS 4	VMS 4	VAX 760	VMS 4						Y		1679	Y	Y	Y	Y	Y	Y	Y	Y	Y
	NTC (National Training Center)	VAX	VMS	UNOS	SUN	UNOS						Y		2167	Y	Y	Y	Y	Y	Y	Y	Y	Y
	OBIEWS (On Board Electronic Warfare Simulator)	750 VAX	VMS 4.4	VMS 4.4	750 VAX 4.4	VMS 4.4	Y	Y	Y	Y				483	Y	Y	Y	Y	Y	Y	Y	Y	Y
	RIRIS (Radiological Release Information System)	VAX 11/750	VMS	VMS	VAX 11/750	VMS						Y	Y	1679	Y				Y	Y		Y	Y
	SCENE (Scenario Generator)	VAX	VMS	VMS	VAX	VMS		Y	Y	Y	Y	Y	Y	1679	Y	Y	Y	Y	Y	Y	Y	Y	Y
	SIMSTAR Preprocessor	VAX	VMS	VMS	VAX	VMS							Y	1679	Y	Y	Y	Y	Y	Y	Y	Y	Y

TABLE 2.1 CANDIDATE PROJECTS AND CHARACTERISTICS - PART 2 OF 2

PROJECTS	Type of Project																	
	BATCH	EVENT CONTROL	PROCESS CONTROL	PROCEDURE CONTROL	NAVIGATION	FLIGHT DYNAMICS	ORBITAL DYNAMICS	MESSAGE PROCESSING	DIAGNOSTIC SOFTWARE	SENSOR/SIGNAL PROCESSING	SIMULATION	DATABASE MANAGEMENT	DATA ACQUISITION	DATA PRESENTATION	DECISION/PLANNING AID	PATTERN/IMAGE PROCESSING	SYSTEM SOFTWARE	DEVELOPMENT TOOLS
AFATDS (Advanced Field Artillery Tactical Data System)			Y	Y				Y	Y	Y	Y	Y					Y	
MC3 (Mideastern C3 Communications Protection Plan)								Y				Y		Y	Y		Y	Y
NTC (National Training Center)		Y						Y	Y	Y	Y	Y	Y	Y	Y		Y	Y
SCENE (Scenario Generator)	Y						Y		Y		Y			Y	Y			

Table 2.2. Representation of STH Software Categories by Chosen Projects

The system contains approximately 72K source lines of FORTRAN 77 code by actual count. This is approximately double the earlier estimate. Although this project is primarily a simulation type of project, it also contains attributes that would qualify it for process control, procedure control, message processing, sensor and signal processing, database management, diagnostic software, and system software. The project was developed and run on a DEC VAX/VMS system. Project documentation is complete except for the detailed design which is documented in a high-level Program Design Language (PDL). The available source code is from Version 2.0, the first fully integrated software build. A description of the tools and methodologies employed during software development is documented in Volume 3 of the Task II Report [8].

Space Defense Scenario Generator (SCENE) - SCENE is an Air Force system that models a variety of sensors and foreign launch missions under the Surveillance Command and Control Design Analysis and Engineering (SCCDA&E) contracts. Its primary purpose is to generate sensor observations of hypothetical foreign launches and range safety operations (RSOs). The sensor's RSO and launch scenarios are user-specifiable.

This software is written in FORTRAN and contains approximately 24K lines of source code. It is a batch type of program that also contains the following functions: orbital dynamics, diagnostic software, simulation, data presentation, and a decision and planning aid. The program runs on a VAX/VMS system. Documentation is complete, although the detailed design specification reflects the software as delivered. The available source code is Release 6.0, prior to system turnover. A description of the tools and methodologies employed during software development is documented in Volume 3 of the Task II Report.

Additional Projects - Two additional projects were available for use: Mid-eastern C3 Communications Protection Plan (MC3) and the National Training Center (NTC) Core Instrumentation Subsystem (CIS). RPFOMs were collected for NTC-CIS in an earlier software reliability study for RADC, thus making it redundant to do so again. MC3 was too large for our present effort budget. As a result, the AFATDS Sim/Stim and SCENE projects were utilized exclusively for the experiments.

2.1.4 Software Project Materials

Basic characteristics of the software projects which were utilized in this study are shown in Table 2.3. Available metric data from the two selected projects is shown in Table 2.4. Supplemental project information (particularly for system application type) is available in Project Characteristics Work Sheets found in Volume 3 of the Task 2 Report. Appendixes A through D of that volume contain matrices of the test project software and associated Software Problem Report (SPR) histories.

Table 2.3. Characteristics of Selected Projects

PROJECTS	Contract Type		Program Size	Environment				MIL-STD Followed
	USAF	DoD		Development		Target		
				HW	SW	HW	SW	
AFTATDS Simulator/ Stimulator (SIM/STIM)		Y	72K	VAX 11/780	VMS 4.2	VAX 11/750	VMS 4.2	None
SCENE (Space Defense Scenario Generator)	Y		24K	VAX	VMS	VAX	VMS	2167

Table 2.4. Availability of Metric Data

METRIC	SIM/STIM	SCENE
Application	X	X
Development Environment	X	X
Anomaly Management		X
Traceability		
Quality Review		X
Language Type	X	X
Modularity	X	X
Complexity	X	X
Standards Review	X	X

2.1.6 Lessons Learned

Several difficulties were encountered while performing the software development project survey and subsequent evaluation of the project materials. They included:

- a. Establishing a thorough survey questionnaire.
- b. Qualifying a knowledgeable project contact for technical information.
- c. Validating the completeness of the available project materials against the questionnaire.
- d. Establishing expedient procedures for evaluating the project materials and for estimating the effort involved.

The following recommendations will resolve each of these cited difficulties:

- a. Revise the survey questionnaire based on known desired information.
- b. Identify the key technical project contact up front and ensure his/her willingness to provide full support.

- c. Personally travel to each project site and verify the availability and applicability of all project materials.
- d. Apply the evaluation methodology that has evolved from this study.

They are recommended for future efforts.

2.2 Testing Techniques Survey

A survey of the capability, advantages, and limitations of many of the static and dynamic software testing techniques currently in use was conducted by reviewing various reference documents and by interviewing test consultants. A list of these reference documents is presented in Appendix A of the Task I Report [5].

2.2.1 Candidate Techniques for Consideration

Software quality should be a primary concern in software development efforts. The traditional methods of assessing software quality are software evaluation and software testing.

Software evaluation examines the software and the processes used during its development to see that its stated requirements and goals are met. Static analysis techniques employ this method of software quality assessment. In these techniques, the requirements and design documents and the code are analyzed, either manually or automatically, without actually executing the code.

Software testing involves actual execution of the program. Dynamic analysis techniques employ this method of software quality assessment. The principal applications of dynamic analysis include program testing, debugging, and performance measurement. This involves the processes of preparing for test execution and analysis of test results.

The following static analysis techniques were surveyed for use in the experiments:

- a. Code Reviews - Assesses conformance of the program implementation to a prepared checklist.
- b. Error and Anomaly Detection - Checks program syntax, coding standards, data and interfaces for anomalies.
- c. Structure Analysis/Documentation - Checks correctness of a program's control and code structures.
- d. Program Quality Analysis - Measures a program's complexity and quality attributes.
- e. Input Space Partitioning - Partitions the program input space into path domains to exercise selected paths.
- f. Data-Flow Guided Testing - Partitions a program flow graph into intervals for analysis of correct sequences of operation.

The following dynamic analysis techniques were surveyed:

- g. Instrumentation-Based Testing - Inserts non-interfering probes into the program to monitor execution behavior and performance.
- h. Random Testing.- Samples the program input domain to find obscure processing errors.
- i. Functional Testing - Finds discrepancies between the program execution and its specification.

2.2.2 Evaluation Criteria

In conducting the survey, the following evaluation criteria were used: usability of the testing technique; use of the technique in the software life cycle; SAIC and RTI projects that utilized the technique; and strengths, weaknesses, and special considerations. Tables 2.5 and 2.6 present an overview of these criteria for each of the testing techniques.

The column entitled USEABILITY summarizes the application of each testing technique and the extent of its usage.

The column LIFE CYCLE identifies the applicable test phase in the software life cycle when the test technique may be employed. The test phases for a program in progressive order are:

- a. Unit Test - The smallest compilable entity is tested.
- b. Unit Integration and Test (CSC Test) - Multiple interfacing units are tested to the component level.
- c. Program Test (CSCI Test) - The entire CPCI is tested as a correct implementation of the specified design; component interfaces are verified.
- d. System Test - The entire System is tested to meet its system level requirements; integrating software with the associated hardware.

Some of these techniques had been applied on specific SAIC/RTI software projects (indicated in the middle column of Table 2.5) as follows:

- a. Code Reviews were conducted through the entire life cycle of SAIC's Performance Analysis and Test (PAT) Program. Peer Reviews audited unit code and formal review tracked the software product across development, design, coding, unit test, CSC test, and program/system test phases of the project.
- b. In providing Independent Verification and Validation (IV&V) support to the Joint Cruise Missile Project Office, SAIC conducted metrics-directed testing of the Mission Planning system (MPS) software for the Air-Launched Cruise Missile (ALCM). An automated testing tool was used to measure software complexity. Additionally, several test covers were developed in order to determine the amount of testing required for each test path. Each test cover provides information on the software complexity value for each module, and indicates which paths are to be exercised with greatest intensity, those to be exercised with less intensity, and those which will be exercised least.

TESTING TECHNIQUE	USEABILITY	LIFE-CYCLE USAGE				SAIC/RTI PROJECTS	STRENGTHS, WEAKNESSES, AND SPECIAL CONSIDERATIONS
		UNIT	CSC	PGM	SYS		
Code Reviews: Peer Reviews	Widely used. Applicable to any size project.	X	X			PAT	Highly effective in detecting errors early in the life cycle. Promotes easier functional system level testing.
Code Reviews: Formal Reviews	Widely used on large, formal or Government projects.	X	X	X	X	PAT	Effective in detecting errors early in the life cycle. Promotes easier functional system level testing.
Error and Anomaly Detection	Extensive usage. Often automated.	X	X				Highly effective in detecting errors early in the life cycle. Automated tools available: PFORT, JAVS, Lint-Plus, MAT, PET, RXVP-80, DAVE, TRAILBLAZER, SOFTOOL-PE, TOOLPACK, SDDL, FAVS, FORTRAN-Lint.
Structure Analysis/ Documentation	Widely used, but limited capability to Detect some errors.	X	X				Effective in debugging. Automated tool: MAT, AMS PET, DAVE, SDDL, FTN-77, JAVS, REFTRAN, FAVS, Source Code Analyzer..
Program Quality Analysis	Measures SW complexity; produces test cover; indicates paths to be tested more intensively; limited use.	X	X	X	X	JCMPO Support	SAIC is a pioneer in, of metrics-directed testing. Supplements the path analysis technique. Tools: MITS, RXVP-80, MMM, AMS, SOFTOOL-PE, MAT.
Input space partitioning: Path Analysis	Detects computation, path selection, and missing path errors; limited use, due to complexity.	X	X			RRJS NTC CIS	Effectiveness restricted by quantities of code "Paths". Tool: RXVP-80.
Input space partitioning: Domain Testing.	Limited usage; detects path selection errors by selecting test data on and near boundary of path domain.	X	X				Not implemented for many programming language features. TOOL: STATUS.
Input space partitioning: Partition Analysis	Examines partitions to determine how closely implementation and specification agree; limited use.	X	X				Still under development as a technique. Reliability is questionable.
Data-Flow guided testing	Used in: optimizing compilers to analyze structural properties of programs; limited use.	X	X				Test strategies are difficult to apply in practice. TOOLS: Available for most languages.

Table 2.5 SUMMARY OF STATIC ANALYSIS TECHNIQUES

TESTING TECHNIQUE	USEABILITY	LIFE-CYCLE USAGE				SAIC/RTI PROJECTS	STRENGTHS, WEAKNESSES, AND SPECIAL CONSIDERATIONS
		UNIT	CSC	PGM	SYS		
Instrumentation - Based Testing: Path and Structural Analysis	Structure-driven technique; very effective in detecting many errors.	X	X				Requires Functional (Requirements) testing as a supplement. TOOLS: JAVS, Lint-Plus, PET, RXVP-80, Softool-PE, STATUS, Tool Pack, FAVS, FORTRAN-Lint, FTN-77, Expeditor, Performance and Coverage Analyzer
Instrumentation Based Testing Performance Measurement - execution time and resource analysis	Widely used in optimization.	X	X	X			Applicable to any kind of program in any programming language. TOOLS: FAVS, SOFTOOL-PE, STATUS, RXVP-80.
Instrumentation Based Test: Performance Measurement Algorithm Complexity Analysis	Limited to highly technical or mathematical applications.	X					Requires much user experience and effort, but highly effective. TOOLS: FTN-77
Instrumentation Based Test: Executable Assertion testing	Limited usage in High Order Languages due to its complexity	X	X				Effective error detector. Complements functional and/or path and structural analysis techniques. TOOLS: RXVP-80, JAVS, FTN-77..
Instrumentation Based Test: Interactive Test and Debug aids.	Widely used, but complex; sometimes used in interactive debugging.	X	X				Tools are language and operating system dependent, and dependent on user efficiency. TOOLS: Lint-Plus, FORTRAN-Lint, FROTREF, FTN-77
Random Testing	Program tested by randomly choosing subset of all possible input values; limited use.	X	X	X	X	LIC	Effective as a supplement to many other techniques. TOOLS: STATUS, TRAILBLAZER.
Functional Testing	Used to find discrepancies between the program and its external specification; widely used.	X	X	X	X	B-1B TSC NTC CIS ADAS PAT	Essentially a manual technique, but is highly effective in finding errors and mapping requirements specifications to program outputs. TOOLS: RTT.

Table 2.6 Summary of Dynamic Analysis Techniques

- c. During both the Radiation Release Information System (RRIS) and the National Training Center (NTC) Core Instrumentation Subsystem (CIS) projects, Path Coverage Testing was used during integration and test of modules.
- d. Random Testing was conducted in NASA projects by RTI during which an automated testing tool was utilized.
- e. Functional Testing was performed during system testing of the B-1B Bomber (B-1B) Technical Support Center (TSC) baseline system and during software testing for the NTC CIS. In both cases, Requirements/Test Matrices were used to determine the specification requirements that were not met during testing. Additionally, the Digital Equipment Corporation (DEC) Test Manager tool was used directly during Functional Testing of the Architectural Design and Assessment System (ADAS) project.

The column STRENGTHS, WEAKNESSES, AND SPECIAL CONSIDERATIONS summarizes the advantages and limitations of each test technique. Complementary characteristics (an advantage gained by using a test technique with one or more others) and available tools and/or methods are also described. Entries in this column are expanded upon below.

- a. Code Reviews (both Peer Reviews and Formal Reviews) are widely used, applicable to large and small projects, are not limited by project type or complexity, and catch errors early in the life cycle. These reviews assist and ease subsequent system level testing, but still require some static testing for complete verification.
- b. Error and Anomaly Detection techniques are highly effective, are extensively automated (but language dependent), and are most applicable during the unit test and CSC test periods of the life cycle.
- c. Structure Analysis/Documentation techniques and tools are widely available and used. This technique's prime utility is in the early stages of debugging, and is most applicable with complex program control flow, but covers only a limited range of programming standards and possible error situations.
- d. Program Quality analysis (also known as Metrics-Directed Testing), can be used both for software reliability analysis and for program Structure Analysis. The latter usage will be in combination with the Path Analysis test technique (which is an element of Input Space Partitioning).
- e. Input Space Partitioning consists of three techniques:
 - 1. Path Analysis (also known as Path Coverage) is recognized to have some drawbacks in that determining the paths of a program and selecting data to execute the chosen paths poses a difficult (if not unmanageable) problem. However, by supplementing this testing technique with other methods, it may be possible to select finite subsets of test data for the chosen paths in order to detect certain type of errors.

2. Domain Testing is limited to simple linear predicates and has difficulty in selecting test cases for a program which has a large number of input variables. It concentrates on path selection errors, which requires other testing methods to be used to test a program thoroughly.
 3. Partition Analysis is more a test support method than a reliability test technique. The specification of a program is assumed to be correct, while in practice, it may be incomplete or contain errors.
- f. Data Flow Guided Testing: Test strategies for this testing technique are more difficult to apply in practice than control-oriented strategies.
 - g. Instrumentation-Based Testing consists of four techniques:
 1. Path and Structural Analysis (Branch) is very effective in detecting data, logic and computation errors and is well complemented by Functional Testing.
 2. Performance Measurement is highly effective in identifying performance problems in a program and is applicable during unit test through CPCI test.
 3. Executable Assertion Testing is effective in determining computational, range, and flow errors in early life cycle testing, but is constrained by its resource- and time-consuming complexity of use.
 4. Debug Aids are also efficient, but complex. Its tools are language and operating system dependent.
 - h. Random Testing possesses desirable and effective features. Generated test data may provide near total branch coverage. Automated tools are readily available to perform random testing.
 - i. Functional Testing techniques are based on state-of-the-art design analysis techniques effective in finding errors. Tools are readily available to perform functional testing.

2.2.3 Selected Testing Techniques

Three static analysis techniques and three dynamic analysis techniques were selected for this study:

STATIC	DYNAMIC
Code Review Error and Anomaly Detection Structure Analysis	Branch Testing Random Testing Functional Testing

Code Review. This static testing technique involves the reading or visual inspection of a program with the objective of finding faults. Test personnel perform

a code review of the sample code units using the appropriate documentation and specifications.

Code reviews conducted consist of code reading and code inspections. Code reading includes the study and evaluation of compiled unit source code line-by-line to evaluate data availability and variable initiation. Code inspections are driven by checklists in order to identify common type of errors, including logic errors, on the unit code.

Error and Anomaly Detection. This static testing technique is applied to detect faults via interface checking, physical units checking, and data flow analysis. Test personnel perform error and anomaly detection on the selected sample code units using the appropriate documentation and specifications.

Interface checking analyzes the consistency and completeness of the information and control flow between units. It checks for and detects:

- a. Incorrect number of arguments.
- b. Data type mismatches between actual and formal parameters.
- c. Data constraint mismatches between actual and formal parameters.
- d. Data usage anomalies.

Physical units checking is performed to detect operations performed on variables of more than one type to determine if necessary conversions have been made.

Data flow analysis identifies paths in which there are variable set-use inconsistencies.

Structure Analysis. This static test technique detects faults concerning the control structures and code structures of FORTRAN, and improper subprogram usage.

Test personnel perform structure analysis on the sample code units using the appropriate documents and specifications. Structure analysis is performed to determine the presence of improper or incomplete control structures, structurally dead code, possible recursion in routine calls, routines which are never called, and attempts to call non-existent routines.

Branch Testing. This testing technique combines static and dynamic techniques and detects failures. The static portion is used to determine test data that force selected branches to be executed. The dynamic portion is used to actually run the code with these test data and then obtain test outputs. Test personnel perform branch testing of the sample code units using the appropriate documentation and specifications.

Test coverage analysis is used to detect untested parts of the program. Output data analysis is applied to detect outputs that deviate from known or specified output values.

Random Testing. This is a dynamic testing technique which tests a unit by randomly selecting a subset of all possible input values. This testing technique

detects failures. Random testing is performed to detect outputs from the random inputs that deviate from known or expected output values. Test personnel perform random testing of the sample code units using the appropriate documentation and specifications.

Functional Testing. This is a dynamic testing technique which finds failures consisting of discrepancies between the program and its specification. In using this testing technique, the program design is viewed as an abstract description of the design and requirement specification. Test data are generated, based on knowledge of the programs under test and on the nature of the program's inputs. The test data are designed to ensure adequate testing of the requirements stated in the specification.

Test personnel perform functional testing of the sample code units using the appropriate documentation and specifications. Functional testing is performed to assure that each unit correctly performs the functions that it is intended to perform. This is accomplished by:

- a. Testing using nominal, extreme, and erroneous input values.
- b. Testing for error detection and proper error recovery, including appropriate error messages.
- c. Testing with data output options and formats.

2.3 Test & Support Tools Survey

Software development/testing tools have formed the foundation of the modern programmer's "workbench". As programming problems have become more complex, increasingly sophisticated tools have been constructed to aid in development and testing. Clearly, in the modern programming environment, useful tools coupled with sound testing practices are key to the realization of reliable software systems.

A survey of many of the software testing tools currently and generally available was conducted. This survey involved a review of the current literature (both professional publications and vendor offerings) describing available software testing tools (see Appendix B). Essentially, these tools may be divided into the following three categories:

- a. Static analysis tools.
- b. Dynamic analysis tools.
- c. Test Support Tools.

Static analysis tools support the static analysis techniques described in 2.2.1. These tools automate all or portions of those test techniques and vary in scope and functionality. A characteristic common to each of these tools is the processing and evaluation of the program source code. They range from systems which simply enforce coding standards to systems which carry out sophisticated structured analysis.

Dynamic analysis tools are used to support the dynamic analysis techniques described in 2.2.1. These tools directly execute the program being tested and perform a wide range of functions including coverage analysis, the generation and evaluation of test data, and the production of run-time statistics.

Test support tools facilitate the testing process and, as such, support all of the testing techniques. They do not normally evaluate or execute the program code. Their general function is to provide database, analysis and documentation support for program requirements, design, code and test cases.

2.3.1 Candidate Tools for Consideration

Twenty-two representative and available tools were surveyed to automate the candidate testing techniques and to support experimentation. Each of the candidate tools which were identified by our survey are categorized into the appropriate test category and test technique. A few of these test tools are applicable to more than one category or technique, and this is identified.

The Static Analysis Tools surveyed for use in the experiments include:

- a. Documentation, Analysis, Validation & Error Detection (DAVE) - Automates Error/Anomaly Detection and Structure Analysis/Documentation.
- b. Maintainability Analysis Tool (MAT) - Automates Error/Anomaly Detection.
- c. FORTRAN-Lint - Automates Error/Anomaly Detection.
- d. Source Code Analyzer (SCA) - Automates Error/Anomaly Detection.
- e. Software Design and Documentation Language (SDDL) - Automates Error/Anomaly Detection.
- f. Automated Measurement System (AMS) - Automates Program Quality Analysis.
- g. Metric Information Tracking System (MITS) - Automates Program Quality Analysis.

The Dynamic Analysis Tools include:

- a. RXVP-80 - Automates Error/Anomaly Detection, Static Structure Analysis/Documentation, Static Path Analysis, Dynamic Path/Structural Analysis, Assertion Checking, and Debugging.
- b. FTN-77 Analyzer - Automates Dynamic Path/Structural Analysis, Performance Measurement, Assertion Checking, and Debugging.
- c. Status - Automates Performance Measurement and Debugging.
- d. Trailblazer - Automates Dynamic Path/Structural Analysis and Performance Measurement.
- e. Lint-Plus - Automates Error/Anomaly Detection, Dynamic Path/Structural Analysis and Debugging.
- f. Path Analysis Tool (PAT) - Automates Dynamic Path/Structural Analysis.
- g. Performance and Coverage Analyzer (PCA) - Automates Dynamic Path/Structural Analysis and Performance Measurement.

The Testing Support Tools Surveyed include:

- a. Requirements Tracing Tool (RTT) - Automates software requirements for functional testing.
- b. Database Management System (DBMS) - Automates program test and data management.
- c. LIBRARIAN - Automates program management.
- d. Code Management System (CMS) - Automates program and module management.
- e. Change and Configuration Control Environment (CCC) - automates program management.
- f. Test Manager (TM) - Automates test management.
- g. Metric Maintenance Manager (MMM) - Automates metrics management.
- h. Statistical Modeling and Estimation of Reliability Functions for Software (SMERFS) - Automates software reliability estimation.

Table 2.7 provides applicable purchase information for each candidate tool, including vendor, price, terms and conditions, version and documentation.

2.3.2 Evaluation Criteria

The following criteria were used to select the necessary test and support tools for the experiments from among the 22 candidates:

- a. Timely availability of the tool.
- b. Familiarity with the tool.
- c. Applicability of the tool to candidate projects.
- d. Applicability of the tool to candidate testing techniques.
- e. Data collection and analysis of software measurements.
- f. Acquisition, generation, and management of test data.
- g. Statistical capabilities for analysis of experiments.
- h. Vendor, host computer and applicable source languages.

2.3.3 Selected Test & Support Tools

The goal of the study was to measure effectiveness of test techniques. Tools were chosen to support the techniques chosen. The experimental design approach taken was to make the effect of the tools as negligible as possible and emphasize the test technique. Comments were gathered from testers and operators on tools at the end of the experiment and empirical study. To decrease impact of a tool effect, the best tool available to support and/or automate a test technique was desired. Inputs from test experts familiar with commercial tools aided the final selection of tools identified in the survey in Task 1. Thus the same tools, the best available within budget, hardware, and operating system constraints, were used by all testers for a given technique. Three software testing tools were selected. These tools are shown in Table 2.8 and are described below.

Table 2.7 TOOLS PURCHASE INFORMATION (Page 1 of 3)

TOOLS	VENDOR	PRICE	TERMS/ CONDITIONS	VERSION/ AVAILABILITY	DOCUMENTATION
Toolpack	Jeff Kram Numerical Algorithms Group 1101 31st St., Suite 100 Downers Grove, IL 60515-1263 (312) 971-2337	\$610 - 1st copy (Public Domain) \$500 - 2nd copy	Public Domain	Release 2.1	Toolpack/1 Introduction Guide (Edition 2) README Guide TIECODE Installers' Guide TIEVMS Installers' Guide TIEC Installers' Guide Tool Installers' Guide
MAT	Gerald Barns SAIC 1213 Jefferson Davis Hwy Suite 150 Arlington, VA 22202 (703) 979-5910	\$4,500 - 1st copy \$3,600 - 2nd copy	One-time perpetual fee, includes 12-month maintenance plan.	Version 10.5	User's Manual
Fortran-Lint	John Dee IPT Corp. 1096 E. Meadow Circle Palo Alto, CA 94304-9284 (415) 494-7500	\$4,500	Perpetually licensed for a one-time fee, on a per CPU basis includes 1 year maintenance.	Version 2.3	Instruction Manual
SCA	Tracy Trent DEC 5471 Kearny Villa Rd., Suite 201 San Diego, CA 92123 (619) 292-1818	\$4,840 ⁽¹⁾	One-time perpetual (?) license fee includes 1 year maintenance period.	Available	User's Manual
SDDL	Don Heinburger SAIC 10760 Campus Point Dr. San Diego, CA 92121 (619) 546-6048	\$5,000	One-time perpetual license fee includes 1 year maintenance period.	Version 3.22	User Instruction Manual Test Case Examples
AMS	Joe Cavano Rome Air Development Center Griffiss Air Force Base, NY 13441 (315) 330-4063	O.P.B.	O.P.B.	First release available Q3/Q4 87	TBD: Available at delivery
MTS	Jermy Smith SAIC 10760 Campus Point Drive San Diego, CA 92121 (619) 546-6222	\$10,000	One-time perpetual fee, includes 1 year maintenance (?)	Available	Technical Data Sheets User's Manual

(1) Estimated

(2) Apparent

(3) Not available at this time

Table 2.7 TOOLS PURCHASE INFORMATION (Page 2 of 3)

TOOLS	VENDOR	PRICE	TERMS/ CONDITIONS	VERSION/ AVAILABILITY	DOCUMENTATION
RXVP 80	Chris Andrews General Research Corp 433 Hillister Ave Box 6770 Berkeley, CA 94701-6770 (805) 964-7724	\$10,000 (Static Analysis, Coverage Analysis, and Program Documentation) or \$4,500/component	Perpetual license includes 90 day maintenance and warranty period.	Available	User's Manual, Sample Report
FTN 77	Louise Grooms National Tech. Info. Ser. Dept. of Commerce 5285 Port Royal Rd. Springfield, VA 22161 (703) 487-4807	\$1,050	Public Domain -Resale restrictions	1985 version	User's Manual, Technical Paper
STATUS	Brian Johnson SALC 10260 Campus Point Drive San Diego, CA 92121 (619) 546-6428	(3)	One time perpetual fee ⁽¹⁾	Available	None
Troubleshooter	Don Courmeir TASC One Jacob Way Reading, MA 01867 (617) 944-6850	\$35,000	One time perpetual fee.	Version 2.0	User and Installation Manual
List Plus	John Dee IFT Corp. 1096 E. Meadows Circle Palo Alto, CA 94304-9284 (415) 994-7500	\$5,500 for all 3 modes (source analysis mode only \$4,500)	Perpetually licensed for a one time fee, on a per CPU basis.	Version 2.5	User Instruction Manual
PAT	Gerald Berns SALC 1213 Jefferson Davis Hwy Suite 150 Arlington, VA 22202 (703) 979-5910	\$2,500 - 1st Copy \$1,750 - 2nd Copy	One time perpetual fee includes 12 month maintenance plan.	Version 3.0	User's Manual
PCA	Tacey Trent DFSC 5471 Kearny Villa Rd., Suite 201 San Diego, CA 92123 (619) 292-1818	\$8,100	(One time perpetual fee includes 12 month maintenance plan ⁽²⁾)	Available	User's manual

(1) Estimated

(2) Agreed

(3) Not available at this time

Table 2.7 TOOLS PURCHASE INFORMATION (Page 3 of 3)

TOOLS	VENDOR	PRICE	TERMS/ CONDITIONS	VERSION/ AVAILABILITY	DOCUMENTATION
RTT	Jenny Smith SAIC 10260 Campus Point Dr. San Diego, CA 92121 (619) 546-6222	\$12,500	One time perpetual fee.	Available	User's Manual
DBMS	Tracy Thern DSC 5471 Kearny Villa Rd. Suite 201 San Diego, CA 92121 (619) 292-1818	QE 899.42 \$22,020	One time perpetual fee.	Available	User's Manual
Librarian	Jenny Smith SAIC 10260 Campus Point Dr. San Diego, CA 92121 (619) 546-6222	\$5,000	One time perpetual fee.	Available	User's Manual
CMS	Tracy Thern DSC 5471 Kearny Villa Rd. Suite 201 San Diego, CA 92121 (619) 292-1818	\$10,500	One time perpetual fee.	Available	User's Manual
CCC	Ken Horn Softool Corp. 8055 W. Manchester Ave. Ste. 310 Playa Del Rey, CA 90293 (213) 827-4641	\$30,000	One time perpetual fee, includes maintenance and training credit.	Version 3.0	Executive Manual system Manager Manual User's Manual Reference Manual
TM	Tracy Thern DSC 5471 Kearny Villa Rd. Suite 201 San Diego, CA 92121 (619) 292-1818	\$9,100	One time perpetual license fee includes 1 year maintenance period.	Version 2.0	User's Manual
MMRM	Jenny Smith SAIC 10260 Campus Point Dr. San Diego, CA 92121 (619) 546-6222	\$10,000	One time perpetual fee.	Available	User's Manual
SMERFS	Joe Cavano Rom Air Development Center Griffiss Air Force Base NY 13441 (315) 330-4063	G.F.E.	G.F.E.	Delivered to SAC	User's Manual Technical Data Sheets

(1) Estimated
(2) Approved
(3) Not available at this time

Table 2.8. Software Test Support Tools

TOOL	APPLICATION	SOURCE
DTM	Manage software samples, test cases, test data.	DEC
SDDL	Support code reading.	SAIC
RXVP-80	Support error and anomaly detection, structural analysis, and branch testing. Also used to provide path coverage information.	GRC

DEC Test Manager. The Dec Test Manager (DTM) is used to organize online test information for the three dynamic testing techniques. Within the DTM testers define test descriptions; each test description defines one test, by associating together the appropriate test sample, its ancillary data files (if any), the sample driver, input case(s), expected outputs (the benchmark), and actual test outputs. One or more test descriptions are organized into a DTM test collection. The DTM allows test execution at the test collection level. All tests for the dynamic test techniques are run as DTM test collections. The DTM automatically stores test outputs and compares them with expected outputs (the benchmark), flagging all mismatches between actual and expected outputs.

SDDL. The Software Design and Documentation Language is used in conjunction with the code review technique only. Neither testers nor operators use the SDDL tool. All code samples were run through the SDDL tool by SAIC personnel prior to the beginning of this study. Each tester received the printed outputs of SDDLs static processing of their code samples. These outputs consist of:

- a. Table of contents for SDDLs output.
- b. Source code, enhanced with indentation based on structure and with flow line arrows in a two-dimensional representation.
- c. Module cross-reference listing.
- d. Module reference tree.

RXVP80. The RXVP80 test tool combines static and dynamic analysis features. RXVP-80 is used to automate structure analysis and error and anomaly detection, to process code samples to identify branch coverage for the branch testing technique, and to instrument samples for path coverage information for all three dynamic techniques. Testers use the static features of RXVP-80 to obtain reports for static analyses of the code samples as required for the static techniques. They instruct RXVP-80 to

instrument the code under test with path coverage commands to create an instrumented version of the source code for use in dynamic testing. These operations are performed by running RXVP-80 independently of the DTM. Subsequently, code samples which are instrumented with the dynamic component of RXVP-80 are invoked from within a DTM test description template file.

3.0 SOFTWARE RELIABILITY PREDICTION

The goal of software reliability prediction is the assessment of a software system's ability to meet specified reliability requirements. The RPFOM is a statement of predicted reliability based on software measurements collected during each phase of software development prior to commencement of CSC integration and testing. The RPFOM contributes to attainment of reliability goals by providing feedback for improving software system design and implementation.

3.1 Technical Approach

The following objectives and constraints were considered when defining the reliability prediction data collection activities for this project:

a. Objectives:

1. Gaining better records of code characteristics in a more usable and realistic form.
2. Developing improved techniques for applying the consequent knowledge to prediction in appropriate confidence settings.

b. Constraints

1. The ability to accumulate data of known validity for new applications.
2. The complexity of the prediction techniques.

There are four levels in which RPFOMs are collected:

- a. Project initiation.
- b. Project development environment.
- c. Requirements and design.
- d. Implementation.

Each of these is described in detail in Section 3.3. Table 3.1 identifies the project data that is collected and used to compute the metrics on which these four RPFOMs are based. These RPFOMs are then computed using original project data from the requirements, design, and implementation phases.

Careful thought was given to the data to be collected and to the data collection procedures. Data for computing the RPFOM measures of software reliability were collected utilizing exacting procedures especially prepared for the experiment. Then the reliability predictions were update at the precise intervals specified during the system development cycle. Special database management software was developed for this application, utilizing 4th Dimension (4D). In addition, the RADC Automated Measurement System (AMS) was utilized to automate the collection of raw measurement data from the software project source code.

Software measurement data comprising the RPFOMs was collected on the software development projects described in Section 2.1.3 based on refinements to applicable worksheets and equations in the SRPEG. The RPFOMs were calculated both for the entire projects and for the unit and CSC integration code samples selected for the

Table 3.1 RPFOM Data Requirements

SRPEG Task Section	Data	Other Input	Data Collection [SRPEG]	
			Form	Procedure
101	Application (A)	System architecture diagram; statement of need; required operational capability; system requirements statement	0	1
102	Development Environment (DE)	Requirements document; Specifications document	1	2
103	System Level Characteristics (S)	S1		
	Requirements and Design Representation Metric (S1)	SA x ST x SQ		
	Anomaly Management (SA)	All system documentation and source code	2	3
	Traceability (ST)	Requirements and design documents with a cross- reference matrix	3	4
	Quality Review Results (SQ)	Requirements document; preliminary design specification; detailed design specification	10	5
	Discrepancies (DR)	Discrepancy reports	5	12
104	Software Implementation Metric (S2)	SL x SM x SX x SR		
	Language Type (SL)	Requirements specification	4	6 and 8
	Modularity (SM)	Module size estimates and source code	4	9
	Complexity (SX)	Source code	4	10
	Standards Review (SR)	Source code	11	11

experiment. See Section 3.6.2 (Table 3.19) for the identification of the software test samples.

Data collection and computational requirements for each RPFOM are summarized in the following sections. Figure 3.1 summarizes the RPFOM data collection process. Some of the computations are automated by the RPFOM Database application software. Reference Section 3.4.3 for details.

3.2 Data Collection Resources

Collection of extensive software measurements and computation of the variety of software project RPFOMs proved to be an intensive computer-based application. A medium- to large-scale computer system with software support tools was essential to automate source code measurement and the ancillary data base management function. All levels of available software test project development specifications, including the unit source code, were utilized to collect the RPFOM metrics. A refined set of metric equations, questions and answer sheets for data collection and RPFOM calculations were extracted from the SRPEG and utilized by assigned data collection personnel. While administrative and technical support personnel largely performed this task, there was a need for a junior level programmer to perform non-automated measurements of the software source code. Each of these resources are discussed in the following sections.

3.2.1 Computer Systems

Two Apple Macintosh II systems (one at each data-collection site, i.e., SAIC and RTI and a Mac SE (at the SAIC site) served as automated workstations for performing RPFOM data management, calculations and report generation. Each Mac II system had the following configuration:

- a. 2 Mbyte random-access memory.
- b. 40 Mbyte hard disk drive.
- c. 800 Kbyte floppy disk drive.
- d. High-resolution monochrome monitor.
- e. Dot-matrix printer.

The SAIC Mac II Workstation was linked to a DEC VAX 8650 on which the AMS automated metrics data-collection tool resides. The RTI Mac II Workstation was linked to a network of DEC MicroVAX 2000 Workstations also containing AMS. The SAIC and RTI VAXes were linked via Arpanet.

3.2.2 Support Software

The Integrated Reliability Management System (IRMS) was the basic support system used in this project. A major capability added to the IRMS during this effort was the ability to download automatic metric analyses from a host (VAX) to a data base management system in IRMS (hosted on a Mac).

Two software support tools were selected to automate major portions of the RPFOM data collection, entry and computation subtasks: AMS and 4th Dimension DBMS. They are described in more detail in the following paragraphs.

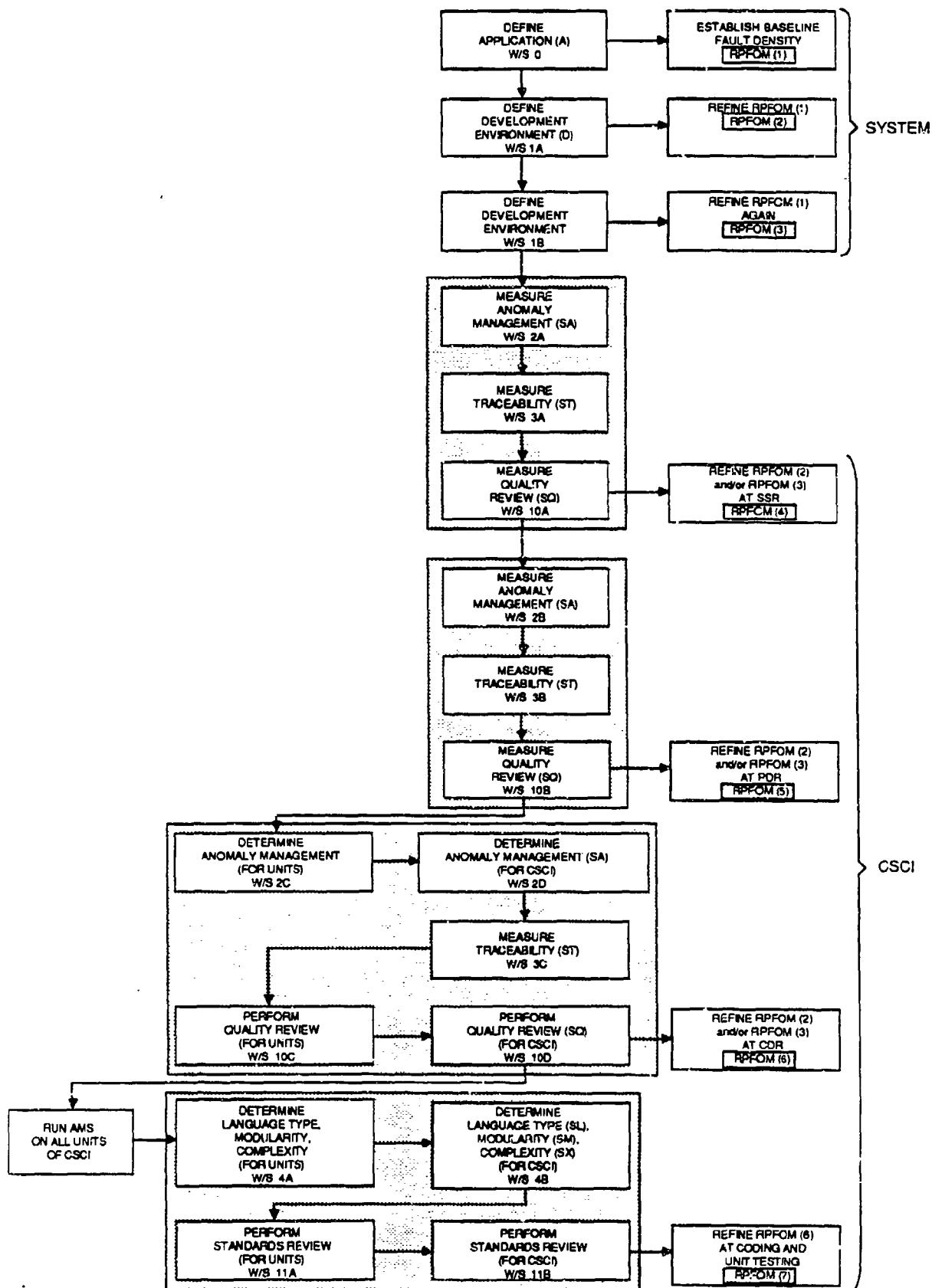


Figure 3.1. Summary RPFOM Data Collection Process Flow

3.2.2.1 Automated Measurement System

AMS reads files of FORTRAN unit source code on the VAX computer system, evaluates 22 of the 34 Standards Review questions in SRPEG worksheet 11A (see Table 2-1), and provides these answers on its own Worksheet E. AMS also automates the collection of many questions on several of the other SRPEG metric work sheets, but requires that the source requirements and design documentation be formalized by use of Software Requirements Engineering Methodology (SREM), Requirements Specification Language (RSL) and the Software Design and Documentation Language (SDDL). As none of these requirements and design specification tools were utilized in our available experimental projects, we were limited to the above mentioned capabilities of AMS Worksheet E.

The Automated Measurement System (AMS) [24, 25] is resident on the DEC VAXes at each data collection site. AMS utilizes worksheets to provide automated measurement of the unit-level metric elements from SRPEG Worksheets 11A and 4A that are listed in Table 3.2. This table also provides the corresponding metric acronyms that appear in the AMS software evaluation report (i.e., Worksheet E) from which the metric values are taken. These values are manually entered into the automated RPFOM Database by the data collector. While AMS was available at both sites, it was utilized exclusively at the SAIC site.

3.2.2.2 4th Dimension DBMS

The 4D DBMS [26, 27, 28] manages files of data on the Apple Macintosh computers. It was a new product which provided user-friendly screen generation and associated relational file definition, creation, maintenance, query, computation and reporting facilities. All of these capabilities were deemed necessary to the development of databases to record, manage and analyze both the experiment RPFOM and software testing results data. The IRMS which is a Mac II based software reliability workstation was previously acquired by RADC under separate contract. Figure 3.2 shows the interfaces between AMS and 4D that were developed for our study.

Our initial approach with the data base management system was to prototype an Entity-Relationship model of the RPFOM worksheets and answer sheets, with corresponding data entry screens and computational procedures. Following this, the complete RPFOM application was developed, including some of the calculations needed for RPFOM computation. Section 3.4.3 describes the level of automated support provided. The user interface to this database is described in Volume 2 of the Task III Report.

3.2.3 Personnel

Contractor experience in applied software reliability measurement test and integration techniques was augmented by contributions of carefully selected consultants with knowledge of the test projects and metric data collection software support tools. Specific personnel roles during the conduct of this task were as follows.

- a. Activity Leader (1). Key project person; responsible for activity assignment, coordination, completion, and products.

Table 3.2. Cross-reference of automated Metric elements from Metric Work Sheet 11A and AMS Software Evaluation DCF Report (Phase E, unit level)

Work Sheet 11A	AMS
MO.1(3)	MO.1(4e)
MO.1(4)a	MO.1(5)e
MO.1(5)	MO.1(7e)
MO.1(7)	MO.1(9e)
MO.1(9)	MO.1(3e)
SI.1(2)	SI.1(2e)
SI.1(3)	SI.1(3e)
SI.1(4)	SI.1(4e)
SI.1(5)a	SI.1(5e)
SI.1(5)b	SI.1(6e)
SI.1(10)	SI.1(11e)
SI.4(1)	SI.4(1e)

Work Sheet 11A	AMS
SI.4(3)a	SI.4(4e)
SI.4(4)a	SI.4(6e)
SI.4(5)	SI.4(8e)
SI.4(6)a	SI.4(9e)
SI.4(8)a	SI.4(11e)
SI.4(9)a	SI.4(12e)
SI.4(9)b	SI.4(13e)
SI.4(10)a	SI.4(14e)
SI.4(10)b	SI.4(15e)
SI.5(3)	SI.5(4e)

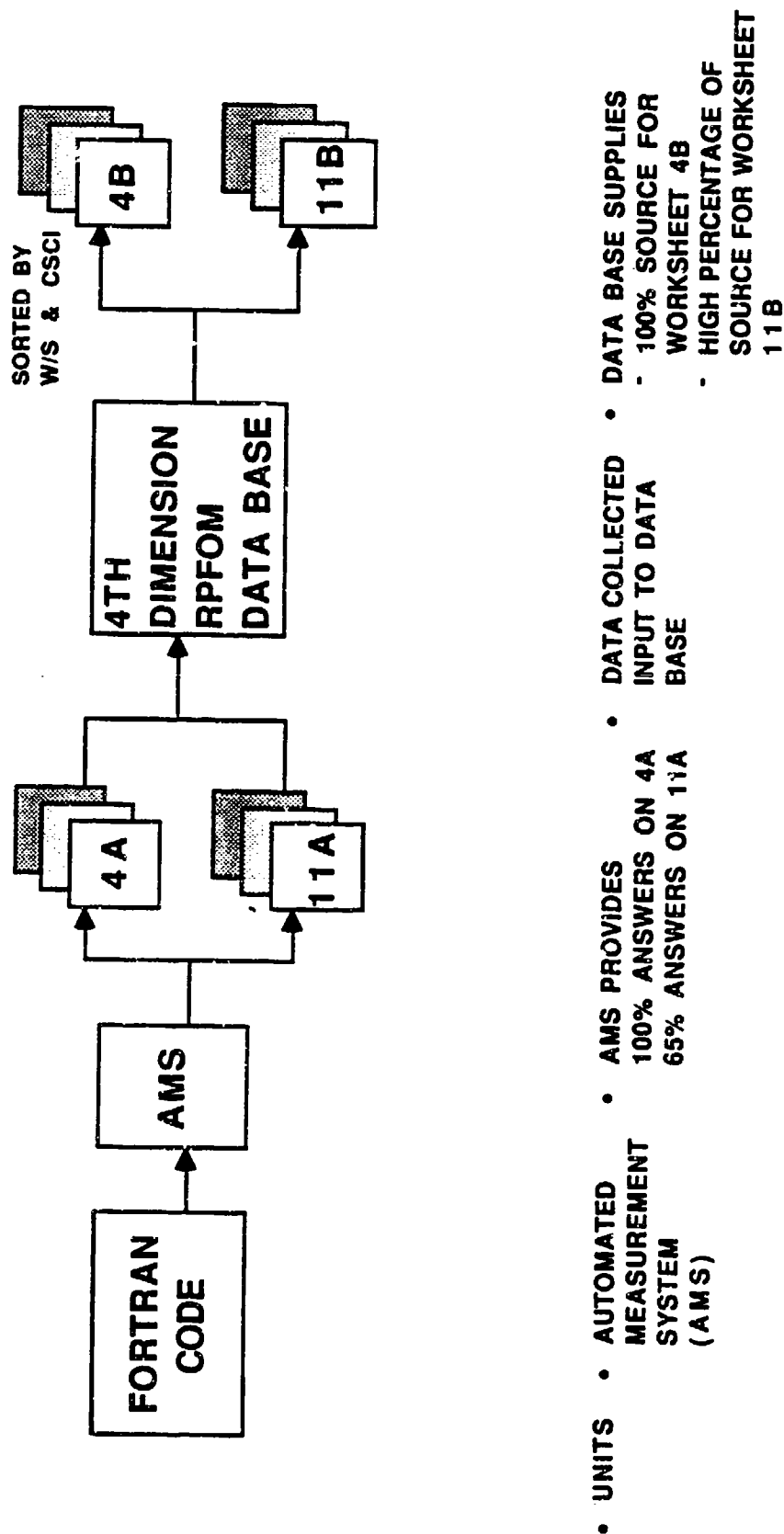


FIGURE 3.2 AMS INTERFACE WITH 4TH DIMENSION

- b. Data Collector (4). Technical support staff and junior engineer; completes data collection forms on a per-project basis; executes metrics collection tool and produces measurement reports.
- c. Data Entry (2). Administrative staff; enters metric data into the database and performs metric computations.
- d. Project Consultant (2). Senior engineer; advises in the acquisition and evaluation of the test project materials.
- e. Software Consultant (2). Engineer; advises in the application and operation of the metrics collection tool (AMS) and the DBMS software (4D).

3.2.4 Instruction Manual

An instruction manual was developed for RPFOM data collection and computation. The purpose of this manual was to provide a concise set of easy to use instructions, based upon the SRPEG and tailored to the objectives of our experiment.

This tailoring involved organizing applicable worksheets of the SRPEG into a set of simplified detail instructions, worksheets and answer sheets that correlate precisely with the sequence of availability of data sources (i.e., documentation and code) that are customarily produced during the Software Development Life Cycle. Table 3.3 shows the correlation between the SRPEG and these worksheets and answer sheets.

A number of useful corrections, clarifications, and refinements are described in Section 3.5 and are incorporated in the Volume 2 guidebook, along with appropriate revisions to the worksheets plus all of the newly developed answer sheets. These changes to the SRPEG and the resulting instruction manual provided the instructions, work sheets, and answer sheets necessary to support RPFOM data collection and computation. Data sources, input and output requirements, and automated tools for data collection and management are described there. A prerequisite to applying these instructions is a breakdown of the project software into CSCIs, CSCs, and units.

3.2.5 Project Documentation

All available metric source documentation was acquired and utilized for each of the software development projects which had been selected for the experiments. Two projects, the AFATDS Sim/Stim and SCENE project were utilized, due to the intensive nature of the reliability prediction data collection process and the available effort for this study. Table 3.4 describes these documents and source tapes.

3.3 Reliability Prediction Methodology

The Reliability Prediction Methodology begins with an architectural diagram. The software components allocated to hardware components can be identified on the system architecture diagram. This allocation should be overlayed on the hardware reliability block diagram. The reliability block diagram shows interdependencies among all elements or functional groups of the system. The purpose of the reliability block diagram is to show by concise visual shorthand the various series - parallel block combinations (paths) that result in successful mission performance. A complete understanding of the system's mission definition and service use profile (operational concept) is required to produce the reliability diagram.

Table 3.3. Association of SRPEG Metric Work Sheets to corresponding Work Sheets and Answer Sheets in RPFOM Instructions

METRIC	SRPEG WORK SHEET	RPFOM INSTRUCTIONS WORK SHEET	RPFOM INSTRUCTIONS ANSWER SHEET
APPLICATION (A)	0	0	1
DEVELOPMENTAL (D1) ENVIRONMENT (D2)	1 1 (checklist)	1A 1B	1 1
ANOMALY (SA) MANAGEMENT	1B1 1B2 1B3 1B4	2A 2B 2D 2C	2 3 5 4
TRACEABILITY (ST)	3	3A 3B 3C	2 3 5
QUALITY REVIEW (SQ)	10A 10B 10C 10D	10A 10B 10D 10C	2 3 5 4
LANGUAGE TYPE (SL)	4	4A 4B	6 7
MODULARITY (SM)	4	4A 4B	6 7
COMPLEXITY (SX)	4	4A 4B	6 7
STANDARDS (SR) REVIEW	11A 11B	11B 11A	TBD 6

3.3.1 Reliability Models

Tasks 101 through 104 of the SRPEG and of the Volume 2 Guidebook provide the procedures for calculating a predictive RPFOM for each component identified in the block diagram according to the following equation:

$$\text{RPFOM} = A * D * S$$

where: RPFOM is the predicted fault density
A is the application type metric
D is the software development environment metric
S is the software characteristic metric

This is Reliability Model 1 as described in the SRPEG. A is expressed in (fractional) faults per line of code, and examples of actual values are presented in Task 101. D and S are modification factors, and each of these can have a value that is less than one (1) if the environment or implementation tends to reduce the fault density. These factors are equivalent to pi factors in MIL HDBK 217E. The Application Area metric represents an average or baseline fault density which can be used as a starting point for the prediction.

Tasks 101 through 104 are preliminary procedures for prediction. Their tables, coefficients, and algorithms are updated in Chapter 5 as a result of data collection and statistical analyses performed on two additional software systems during the present experiments.

For specified software components, a detailed model based on a functional flow analysis can be developed. This Reliability Model 2 was not evaluated in the present study.

3.3.2 Reliability Computations

The functional definition of RPFOM for Reliability Model 1 is:

$$\text{RPFOM} = \text{Predictive Fault Density} = \text{Faults/LOC} \quad (\text{Eq. 1-1})$$

where Fault is a condition that causes a functional unit to fail to perform its required function, and LOC is executable line of code. The RPFOM can be computed for an entire software system and/or its components (e.g., CSCIs). The RPFOM can be converted to a failure rate or calculated as a failure rate also. Four successive RPFOM equations, each representing a refinement to its predecessor, are given:

$$\text{RPFOM} = A \quad (\text{Eq. 1-2})$$

where "A" is a metric for System Application Type;

$$\text{RPFOM} = A * D \quad (\text{Eq. 1-3})$$

where "D" is a metric for System Development Environment;

$$\text{RPFOM} = A * D * S1 \quad (\text{Eq. 1-4})$$

**Table 3.4. List of Project Documentation Available
for Metrics Data Collection**

PROJECT	METRIC TYPE	METRIC WORK SHEET	SPECIFICATION NAME
AFATDS SIM/STIM	APPLICATION (A)	0	A. System Specification for AFATDS, Simulator/ Stimulator Draft 1 April 1986 B. AFATDS Simulator/ Stimulator, Software Overview Final February 10, 1987 C. Subcontractor Statement of Work Revision F May 26, 1986
	DEVELOPMENT ENVIRONMENT (D)	1	Refer to A D. PDL Tapes
	ANOMALY MANAGEMENT (SA)	2A, 2B, 2C, 2D	Refer to A, D
	TRACEABILITY (ST)	3A, 3B, 3C	Refer to A, D
	QUALITY REVIEW RESULTS (SQ)	10A, 10B, 10C, 10D	Refer to A, D
	LANGUAGE TYPE (SL)	4A, 4B	E. Source Code, AMS Work Sheet E
	MODULARITY (SM)	4A, 4B	Refer to E
	COMPLEXITY (SX)	4A, 4B	Refer to E
	STANDARDS REVIEW (SR)	11A, 11B	Refer to E

**Table 3.4. List of Project Documentation Available
for Metrics Data Collection (Continued)**

PROJECT	METRIC TYPE	METRIC WORK SHEET	SPECIFICATION NAME
MC3	APPLICATION (A)	0	A. Command Center System 15-May-1985 B. Program Performance Specification for the RSNFC ³ Program
	DEVELOPMENT ENVIRONMENT (D)	1, 1A	Refer to Specifications A, B C. Program Design Specification for the RSNF C ³ 4-OCT-1985
	ANOMALY MANAGEMENT (SA)	2A, 2B, 2C, 2D	Refer to Specifications B, C
	TRACEABILITY (ST)	3A, 3B, 3C	Refer to Specifications A, B, C
	QUALITY REVIEW RESULTS (SQ)	10A, 10B, 10C, 10D	Refer to Specifications A, B, C
	LANGUAGE TYPE (SL)	4A, 4B	D. Source Code, AMS Work Sheet "E"
	MODULARITY (SM)	4A, 4B	Refer to Specification D
	COMPLEXITY (SX)	4A, 4B	Refer to Specification D
	STANDARDS REVIEW (SR)	11A, 11B	Refer to Specification D

**Table 3.4. List of Project Documentation Available
for Metrics Data Collection (Continued)**

PROJECT	METRIC TYPE	METRIC WORK SHEET	SPECIFICATION NAME
SCENE	APPLICATION (A)	0	A. Space Defense Simulator Program Development Plan 30-July-1984 B. SPADCCS Space Defense Simulation User's Manual 22-April-1987 C. SPADCCS Scenario Generator (Scene) Detailed Design Document Versions 1 and 2 17-September-1984
	DEVELOPMENT ENVIRONMENT (D)	1, 1A	D. Scene Engineering Manual 24 April-1987 Refer to Specification C
	ANOMALY MANAGEMENT (SA)	2A, 2B, 2C, 2D	Refer to Specifications C, D
	TRACEABILITY (ST)	3A, 3B, 3C	Refer to Specifications C, D
	QUALITY REVIEW RESULTS (SQ)	10A, 10B, 10C, 10D	Refer to Specifications C, D
	LANGUAGE TYPE (SL)	4A, 4B	E. Source Code and AMS Work Sheet "E"
	MODULARITY (SM)	4A, 4B	Refer to Specification E
	COMPLEXITY (SX)	4A, 4B	Refer to Specification E
	STANDARDS REVIEW (SR)	11A, 11B	Refer to Specification E

**Table 3.4. List of Project Documentation Available
for Metrics Data Collection (Continued)**

PROJECT	METRIC TYPE	METRIC WORK SHEET	SPECIFICATION NAME
NTC	APPLICATION (A)	0	A. Requirements Design Specification Vol. 1 Sections 1.2, 1.3, 1.4, 1.5, 2.0, 3.1 12 August 1985 NTC-1221-18
	DEVELOPMENT ENVIRONMENT (D)	1	Refer to A, Sections 3.1, 3.2 B. Requirements Design Specification, Vol. II Sections 4.0, 5.0, 16 Sept 83, NTC-1221-18
	ANOMALY MANAGEMENT (SA)	2A, 2B, 2C, 2D	Refer to A, Section 3.2 Refer to B, Sections 4.0, 5.0
	TRACEABILITY (ST)	3A, 3B, 3C	Refer to A, Sections 3.1, 3.2 Refer to B, Sections 4.0, 5.0 C. Accept. Test Plans, 15 Feb 84, NTC-1252-62
	QUALITY REVIEW RESULTS (SQ)	10A, 10B, 10C, 10D	Refer to A, Section 3.2 Refer to B, Sections 4.0, 5.0
	LANGUAGE TYPE (SL)	4A, 4B	D. Source Code E. AMS Work Sheet "E"
	MODULARITY (SM)	4A, 4B	Refer to Specification D, E
	COMPLEXITY (SX)	4A, 4B	Refer to Specification D, E
	STANDARDS REVIEW (SR)	11A, 11B	Refer to Specification D, E

where "S1" is a metric of Software Characteristics during software Requirements and Design Specification;

$$\text{RPFOM} = A * D * S2 \quad (\text{Eq. 1-5})$$

where "S2" is a metric of Software Characteristics during software Implementation. "A" is expressed as a baseline fault density, whereas "D," "S1," and "S2" are modification factors for which values can range from <1 (decreased fault density) to >1 (increased fault density).

The "S1" metric is derived from "SA" (Anomaly Management), "ST" (Traceability), and "SQ" (Quality Review) metrics:

$$S1 = SA * ST * SQ \quad (\text{Eq. 1-6})$$

The "S2" Metric is derived from "SL" (Language Type), "SM" (Modularity), "SX" (Complexity), and "SR" (Standards Review) metrics:

$$S2 = SL * SM * SX * SR \quad (\text{Eq. 1-7})$$

3.3.2.1 Application RPFOM

The Application-type RPFOM (Eq. 1-2) is calculated for each test project. This baseline RPFOM, determined prior to initiation of software development, is an average fault density based on the principle application type of the test project (e.g., Airborne System). Metric Worksheet 0 provides a list of six application types for selection.

3.3.2.2 Development Environment RPFOM

The Development Environment RPFOM, which is a refinement of the baseline RPFOM, incorporates information, summarized in the "D" metric, should be available during a software pre-development phase of the life-cycle. Although this RPFOM is defined by a single expression (Eq. 1-3), one or two worksheets (either 1A, or 1A in combination with 1B) can be utilized to compute "D" depending on the level of detail of project environment data available. In order to evaluate the general applicability of each of these work sheets, Development Environment RPFOM was calculated twice for each test project.

Worksheet 1A provides a quick approximation of "D" based upon selection by the data collector of one of three development environment categories. Worksheet 1B provides a more precise determination of "D" based upon a checklist of 38 development environment characteristics:

$$\begin{aligned} D &= (0.109D_c - 0.04) / 0.014 && \text{if Embedded from W/S 1A} \\ &= (0.008D_c + 0.009) / 0.013 && \text{if Semi-detached from W/S 1A} \\ &= (0.018D_c - 0.003) / 0.008 && \text{if Organic from W/S 1A} \end{aligned}$$

where $D_c = (\# \text{ characteristics in W/S 1B not applicable to system})/38$.

3.3.2.3 Requirements and Design RPFOM

This RPFOM (Eq. 1-4), a refinement of the Development Environment prediction, incorporates information on software characteristics provided by system

requirements and design documentation to determine the SA, ST, and SQ metric components of S1 (Eq. 1-6). Any of three sets of worksheets, each set specific to a particular Life-cycle phase, is used to derive these three metric components. Three Requirements and Design RPFOM values corresponding to SSR, PDR, and CDR were determined for each CSCI of the two software test projects in order to evaluate the usability of each set of worksheets. Derivation of SA, ST, SQ is summarized below.

Anomaly Management (SA): The "SA" metric is equated to one of three values based on the value of "AM," which is derived from responses by data collectors to questions in Worksheets 2A (SSR), 2B (PDR), and 2C/2D (CDR) that apply to the capabilities of a system to respond to software errors and other anomalies:

$$AM = \text{Number of "NO" responses} / \text{Total number of "YES" and "NO" responses}$$

"SA" is then computed automatically as follows:

$$\begin{aligned} SA &= 0.9 \text{ if } AM < 0.4 \\ &= 1.0 \text{ if } 0.6 \geq AM \geq 0.4 \\ &= 1.1 \text{ if } AM > 0.6 \end{aligned}$$

Traceability (ST): A value for "ST" is selected by the data collector using Worksheets 3A (SSR), 3B (PDR), or 3C (CDR) which encompass traceability of requirements from system level through unit level.

Quality Review (SQ): The "SQ" metric is equated automatically to one of two values:

$$\begin{aligned} SQ &= 1.1 \text{ if } DR / \text{Total \# Y and N responses} > 0.5 \\ &= 1.0 \text{ if } DR / \text{Total \# Y and N responses} \leq 0.5 \end{aligned}$$

DR is a count of "NO" responses from Worksheet during Software Requirements Analysis 10A (SSR), 10B during Preliminary Design (PDR), or 10C/10D during Detailed Design (CDR).

3.3.2.4 Implementation RPFOM

The Implementation RPFOM (Eq. 1-5), which represents a final refinement to the reliability prediction at the CSCI level, incorporates information on software characteristics derived from source code during Coding and Unit Testing (C&UT) to determine the SL, SM, SX, and SR components of S2 (Eq. 1-7). Unit-level metrics are collected for Worksheets 4A and 11A, and then summed for corresponding CSCIs using Worksheets 4B and 11B. Data collection for this RPFOM begins with the utilization of the AMS on the VAX for collection of many of the unit-level metric elements. The values obtained from the AMS hard-copy report are then transferred to an answer sheet along with values for non-automated metrics as indicated in the worksheets. Derivation of SL, SM, SX, and SR components of S2 is summarized below.

Language Type (SL): The "SL" metric is derived as follows:

$$SL = (HLOC/LOC) + (1.4 ALOC/LOC)$$

where: HLOC = higher-order-language line of code for CSCI
 ALOC = assembly language lines of code for CSCI
 LOC = total executable lines of code for CSCI

Values for HLOC, ALOC and LOC are determined in Worksheet 4B based upon unit-level values for these variables Worksheet 4A.

Complexity (SX): The "SX" metric is derived as follows:

$$SX = (1.5a + b + 0.8c)/NM$$

where: a = # units in CSCI with $sx \geq 20$
b = # units in CSCI with $7 \leq sx < 20$
c = # unit in CSCI with $sx < 7$
NM = # units in CSCI
sx = complexity for unit

Values for a, b, and c are determined in Worksheet 4B based upon unit-level data provided in Worksheet 4A.

Modularity (SM): The "SM" Metric is derived as follows:

$$SM = (0.9u + w + 2x)/NM$$

where: u = # units in CSCI with $MLOC \leq 100$
w = # units in CSCI with $100 < MLOC \leq 500$
x = # units in CSCI with $MLOC > 500$
MLOC = lines of code in unit
NM = # units in CSCI

Values for u, w, and x are determined in Worksheet 4B based upon unit-level data provided in Worksheet 4A.

Standards Review (SR): The "SR" metric is derived as follows:

SR = 1.5 if $DF \geq 0.5$
= 1.0 if $0.5 > DF \geq 0.25$
= 0.75 if $DF < 0.25$

where: $DF = (\# \text{ "No" responses}) / (\# \text{ "No" } + \text{ "Yes" responses})$

Values for # "No" responses and # "yes" responses are determined in Worksheet 11B based upon unit-level data provided in Worksheet 11A.

3.3.3 Software Reliability Prediction

The results of using Reliability Model 1 is a prediction of software reliability for each block in the system/hardware block diagram. A description of the format and documentation required for a block diagram is in MIL-STD 756B, Task Section 100. The software reliability prediction numbers should be entered on the block diagram and incorporated into the mathematical model of that diagram. The use of these procedures and assumptions made should be documented under paragraph 2.3.8.1, Software Reliability Assumptions, in that task section.

When using Model 1, the predicted software reliability figure of merit is a fault density as described above. The predicted software reliability figure of merit is a

probability that the software will not cause failure of a mission for a specified time under specified conditions. The probability of failure for a specified period of time is given by the failure rate, the expected (average) number of failures per unit time, usually taken as a computer-or CPU-hour. Because the failure rate has a direct correspondence to the definition of software reliability, it is selected as the primary unit of measure for software reliability.

The fault density predicted by Model 1 is used as an early indicator of software reliability based on the facts that: (1) the number of problems being identified and an estimate of size are relatively easy to determine during the early phases of a development and (2) most historical data available for software systems support the calculation of a fault density, but not failure rate. Fault density is the number of faults detected (or expected to be detected) in a program divided by the number of executable lines. Fault density was found to range from 0.005 to 0.02 in high quality software, in early research on software reliability. The prediction of fault density is suitable for the early stages of software development. As information about the intended execution environment becomes available, the predicted fault density can be translated into a predicted failure rate.

The fault density cannot be used directly in the system block model. Instead it can be used as an indicator for unreliable components or critical reliability components. The fault density derived by the prediction methods can be compared to Table 3.5 which contains industry averages or with the specified fault density requirement, if stated in the Requirement for Proposal (RFP). Actions can then be taken in the early phases of development to remedy pinpointed unreliable components through redesign, reimplementation or emphasis and rework during test.

APPLICATION TYPE	TRANSFORMATION RATIO
AIRBORNE	6.2
STRATEGIC	1.2
TACTICAL	13.8
PROCESS CONTROL	3.8
PRODUCTION CENTER	23
DEVELOPMENTAL	NOT AVAILABLE
AVERAGE	10.6

Table 3.5 Transformation for Fault Density to Failure Rate

3.4. Data Collection Framework

The RPFOM data collection process should be applied to software projects during their development. In the present study it was necessary to apply these procedures to completed projects in a manner which emulated their development. Only project data sources which (would have) exist(ed) at the software life-cycle phase corresponding to the metrics of interest were referenced for each RPFOM. This careful attention to the timeliness of all data sources was necessary in order to meaningfully apply and evaluate the reliability prediction methodology in the present study.

3.4.1 General Procedures

General procedures that supported the RPFOM data collection activities are described in the following task sections of the SRPEG:

- a. Task Section 101: pp. TS-9-TS-10: Software Reliability Prediction based on Application.
- b. Task Section 102: pp. TS-11-TS-13: Software Reliability Prediction based on Development Environment.

- c. Task Section 103: pp. TS-14-TS-17: Software Reliability Prediction based on CSCI Level Software Characteristics.
- d. Task Section 104: pp. TS-18-TS-20: Software Reliability Prediction based on CSCI/Unit Level Characteristics.

3.4.2 Life-Cycle Phase Worksheets

Detailed worksheets for each task procedure can be found in the body of the task section. Refined task worksheets and answer sheets as actually used in the experiment are contained in the instruction manual. These worksheets were extracted directly from the SRPEG, with the goal of simplifying their organization, clarifying many of the individual worksheet questions, and resolving several points of inconsistency in worksheets and equations. These refinements are discussed in Section 3.5 and are incorporated into Tasks 101 through 104 of the Volume 2 guidebook.

The instruction manual was organized as a stand-alone document for use by data collectors. Metric worksheets and answer sheets are arranged there in the sequence of their utilization. Associated detailed instructions also appear there. The RPFOM data collected for each test project is specified in the metric worksheets. Each work sheet targets a specific metric(s) (e.g., Quality Review), software life-cycle phase (e.g., Detailed Design), and software component level (i.e., System, CSCI, or Unit) as illustrated in Table 3.6. The collected data can be recorded manually on metric answer sheets prior to entry into the RPFOM Database, as was done during the present study. Answer sheets were prepared to support all worksheets corresponding to a specific software component level and life-cycle phase.

These worksheets and answer sheets were used as directed. Then the collected answers were entered into the RPFOM Database.

3.4.3 Database Development

The RADCS software prediction methodology requires the collection of a significant amount of measurement data from software project documents and source code. Further, the necessary computations of these raw metrics and their summarization into meaningful system and CSCI-level RPFOMs is itself significant. Such could not be economically accomplished without the support of an automated DBMS. It was determined early that a DBMS-based application package was needed which would:

- a. Provide data entry capability for every metric answer sheet in the instruction manual.
- b. Automatically perform as many of the computations on the corresponding worksheets as our effort and schedule budget would permit.
- c. Permit the user to enter answers from any remaining computations on the worksheets.
- d. Retain all original data and computed answers.

Table 3-6. Framework for application of Metric Worksheets (designated by "W/S")

LIFE-CYCLE PHASE		SOFTWARE PRE-DEVELOPMENT		SOFTWARE REQUIREMENTS ANALYSIS	PRELIMINARY DESIGN	DETAILED DESIGN	CODING AND UNIT TESTING	METRIC (ACRONYM)
		SRR	SRR	SSR	PDR	CDR		
Application Level	REVIEW	RPFOM		A x D x S1 (S1 = SA x ST x SQ)			A x D x S1 x S2 (S2 = SL x SM x SX x SR)	APPLICATION TYPE (A)
	Source Documents	A	A x D					
SYSTEM	SSS	W/S 0						DEVELOPMENT ENVIRONMENT (D)
CSCI	SDP		W/S 1A or 1B					ANOMALY MANAGEMENT (SA)
	SRS			W/S 2A	W/S 2B	W/S 2D		TRACEABILITY (ST)
	STLDD					W/S 2C		QUALITY REVIEW (SQ)
	SDDDD							
UNIT	SRS			W/S 3A	W/S 3B	W/S 3C		
CSCI	STLDD							
	SDDDD							
	SDDDD			W/S 10A	W/S 10B	W/S 10D		
	SDDDD					W/S 10C		
UNIT	CODE							
CSCI	CODE						W/S 4A	LANGUAGE TYPE (SL) MODULARITY (SM) COMPLEXITY (SX)
	CODE						W/S 4B	LANGUAGE TYPE (SL) MODULARITY (SM) COMPLEXITY (SX)
UNIT	CODE						W/S 11A	STANDARDS REVIEW (SR)
CSCI	CODE						W/S 11B	

- e. Automate as many combinations of system and CSCI-level RPFOMs from the SRPEG and instruction manual as are meaningful for subsequent evaluation.
- f. Perform interactive computation of user-selected RPFOMs and retain the results.

The resulting RPFOM database is a close model of the RPFOM worksheets/answer sheets and it implements the natural entity relationships between them. This data base was implemented within the IRMS. Figure 3.3 illustrates the RPFOM database relational structures. These structures implement the various worksheets and answer sheets in Table 3.3. They also parallel the system, CSCI and unit hierarchy of each software project in order to record applicable metric data. Volume 2 of the Task III report provides definitions for all of the data items in these structures.

All RPFOMs and a few of the calculations on raw metric data are performed automatically (i.e., S₁, S₂, A, D₁, D₂, SL, SM, SX and SA). The remaining calculations must be computed manually and the results entered into the database. We were not able to automate as many of the basic computations as originally planned due to budget constraints but the potential for doing so exists.

In summary, the RPFOM database is comprehensive and fully automates the RPFOM data management functions for Tasks 201 through 204, excluding the calculations for Worksheets 2C and 10C. These can be added in future experimentation. The database is user friendly, as it is based on the standard Mac-user interface through 4D.

Primary areas to consider during future software reliability experimentation are automation of the remaining Worksheet 2C and 10C calculations, as noted above, and the interface between the RPFOM Database and AMS which feeds it. Right now the latter is a manual interface and need not be. The majority of SRPEG Worksheet 11A is derived presently from the AMS Worksheet E. New effort can be directed to automating this interface and to enhancing AMS to automatically provide the remaining Worksheet 11A metrics while it processes software project source code. Additional candidates for such enhancement are SRPEG Worksheets 2, 3 and 10 (all phases) which could be similarly interfaced to equivalent AMS Worksheets. This will accommodate software development projects which utilize SREM, RSL and SDDL to formalize these metric inputs to AMS.

3.5 Refinements to the SRPEG

Refinements made to the SRPEG during this study can be classified into global changes, worksheet changes and equation changes. They are made in Volume 2.

a. Global Changes

1. All occurrences of "NA" were deleted for components of metric questions which require a numeric response. If a value cannot be determined for a numeric question, the corresponding item on the answer sheet (and in the data base) should be left blank. All other "NA" responses for non-numeric questions should be addressed as being "Not Applicable to Sample Project."

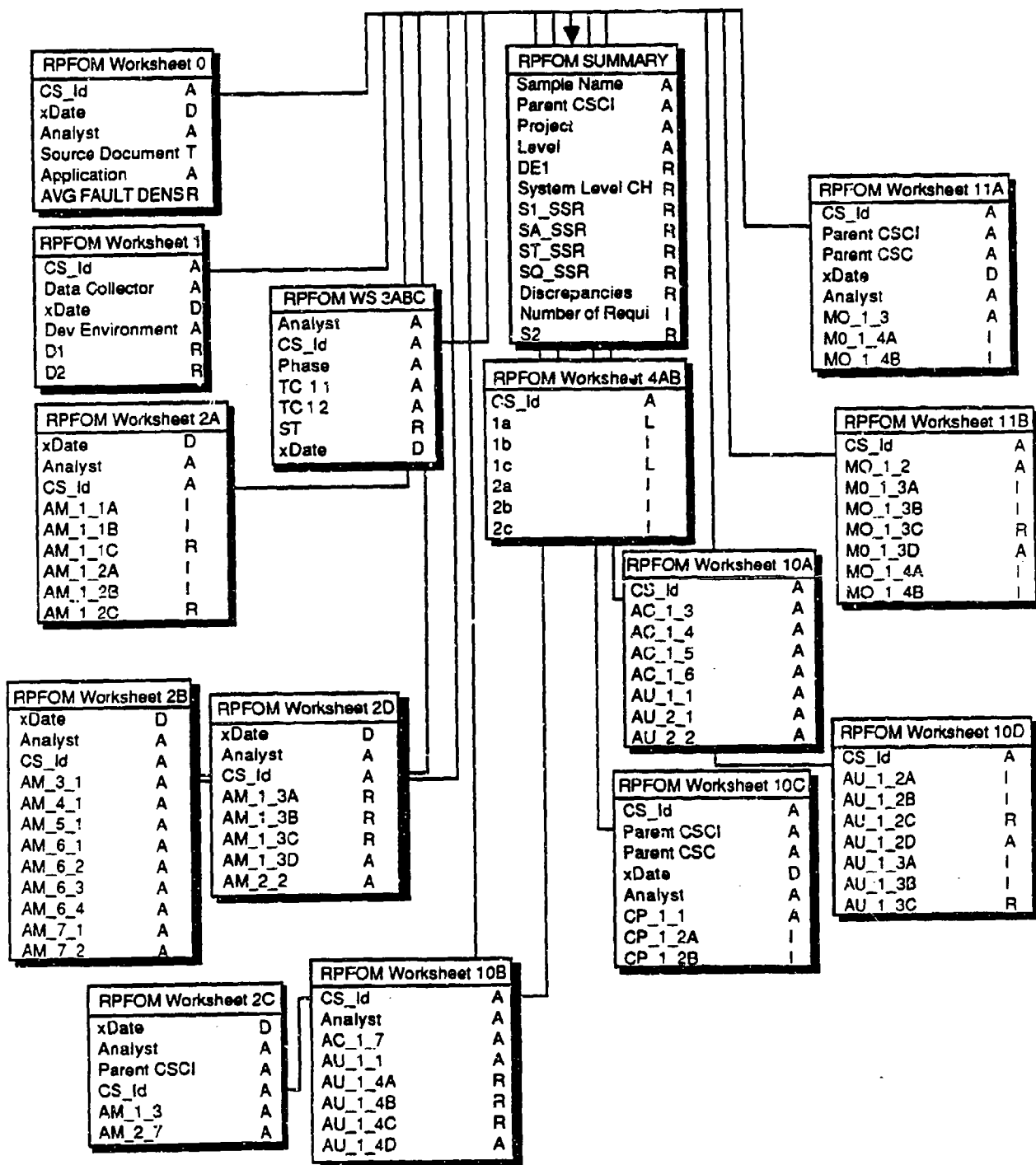


Figure 3.3 RPFOM Database Relational Structures

2. An option of "UNK", or unknown, has been added to the answer sheets and data base for most of the non-numeric metric questions. This is used when a "Yes" response may be warranted, but cannot be chosen with certainty due to the unavailability of information.
3. Questions 1G and 1H on Worksheet 1B, Developmental Environment, were vague regarding average levels for both education and experience. This was resolved by placing the average experience level at three years and the average educational level at a bachelors degree.
4. Worksheets for Standards Review were established for data collection at the CSCI level, Unit level, and CSC levels. It was determined that only the CSCI and Unit level worksheets were appropriate.
5. For Quality Review Results there was an issue of whether we could use AMS answered questions for this phase of the life cycle. It was determined that these three metrics should be based solely on documentation as a data source, since source code does not exist at this point in the life cycle.
6. The size requirements for a unit set by the SRPEG seemed to be excessively large for the modularity metric. Unit size set by the SRPEG list a small unit as under 200 lines of code, a medium size unit had between 200 and 3000 line of code, and a large unit was over 3000 lines of code. Our changes resulted in lowering the size of a small unit to under 100 lines of code, a medium size unit has between 200 and 500 lines of code, and a large unit is over 500 lines of code.

b. Worksheet Changes

1. Worksheets 2A, 2B, 2C, 2D names were changed as follows:

From: Checklist 1B1	To: Worksheet 2A
Checklist 1B2	Worksheet 2B
Checklist 1B3	Worksheet 2D
Checklist 1B4	Worksheet 2C
2. Worksheet 2A, Question AM.1(4) - The word 'exist" was omitted in the question. Both possible responses were "no." One of these was changed to "yes."
3. Worksheet 2B, Question AM.6(1) - Ambiguity in this question was corrected. The duplicate CP.1(1) question was deleted.

5. Worksheet 11A - Only the following questions are answerable utilizing AMS Worksheet E as source:

MO.1(3)	SI.1 (5a)	SI.4(4a)	SI.4(10b)
MO.1(5)	SI.1(5b)	SI.4(5)	SI.5(3)
MO.1(7)	SI.1(5c)	SI.4(6a)	
MO.1(9)	SI.1(5d)	SI.4(8a)	
SI.1(2)	SI.1(10)	SI.4(9a)	
SI.1(3)	SI.4(1)	SI.4(9b)	
SI.1(4)	SI.4(3a)	SI.4(10a)	

Consequently, only corresponding questions in Worksheet 11B can be answered. The equation remained the same but the total number of responses are lessened to "23."

6. Worksheet 11B - The equation at the end of the work sheet asks for no/(no + yes) and assigns this value to DR. However, none of the questions in this work sheet gave a yes/no response. We evaluated all of the questions inserted an equation that would give a valid yes/no response into the questions noted below:

MO.1(5)	SI.1(2)	SI.4(1)
MO.1(3)	SI.1(3)	SI.4(5)
MO.1(6)	SI.1(4)	SI.4(12)
MO.1(7)	SI.1(5)	SI.4(13)
MO.1(8)	SI.2(1)	SI.5(3)
MO.1(9)		

The following additional changes were made to specific questions on Worksheet 11B:

MO.1(9)	This question did not have a corresponding question in 11B. An appropriate question with yes/no conclusion was inserted.
MO.2(2),(3)	Both are hardware questions and were consequently deleted.
SI.4(9)c.	An omitted minus sign was restored in the calculation.

c. Equation Changes

1. Development Environment equations (SRPEG: TS-11; RDCI: 2-5):

SRPEG: $D_C = \text{No. of methods and tools applicable to system (i.e. "yes")}/38.$

New Equation: $D_C = \text{No. of methods and tools not applicable to system (i.e., "no")}/38.$

Anomaly: The SRPEG Dc results in a larger fault-density multiplier correlated with an increase in number of beneficial methods and tools used.

2. Anomaly Management - SA (SPREG: TS-13; RDCI: 2-5):

SRPEG: SA = 0.9 if AM > 0.6
= 1.1 if AM < 0.4

New Equation: SA = 0.9 if AM < 0.4
= 1.1 if AM > 0.6

Anomaly: An increase in AM (proportion of "no" responses) corresponds to a decline in the SRPEG SA (fault-density multiplier), thus incorrectly effecting a higher predicted reliability.

3.6 Data Collection Results

The results of this study classify into two major subjects:

- a. Effort expended on each study activity and on the individual work sheets for each test project.
- b. RPFOM Numbers computed for each test project: first as complete software systems, then for each test sample at the PDR and Code and Unit Test (C&UT) phases.

Findings and Conclusions are discussed in Section 3.7 and are based on our utilization of available resources, application of refined RPFOM worksheets and equations, and conduct of the required study activities.

Analysis and recommendations regarding the RPFOM metric multiplier coefficients are provided in Chapter 5.

3.6.1 Effort Summary

Table 3.7 provides a summary of the combined effort expended by project personnel in the performance of the study activities. These numbers are estimates and are based on project records and notes.

ACTIVITY	MANWEEKS	PERCENT OF EFFORT
AMS Familiarization	4	6
4D Familiarization	4	6
Instruction Preparation	21	30
Database Development	10	14
Material Set-up and Control	3	4
Orientation and Training	3	4
Data Collection and Calculation	25	36
Task III Total	70	100

Table 3.7 Task III Activity Effort Summary

The products of this work for future RADC experimentation in software reliability prediction can be categorized into:

- a. Documentation and source code for four software development projects to be maintained in the RADC software reliability project repository for future experimental use.
- b. Recorded effort data for two of these projects (Sim/Stim and SCENE) upon which to estimate future efforts to measure and predict software reliability.
- c. Raw measurement data from these two projects to incorporate into the RADC software reliability data base for future experimental analyses.
- d. Computed software reliability data from these project test samples for comparison with software testing results.
- e. Simplified instructions for software reliability prediction, and experience gained - both a basis for recommendations and revisions to the SRPEG.

- f. Enhancements to the IRMS utilizing a modern DBMS. The enhancements included an automated SRPEG worksheet management and RPFOM computation capability.
- g. An IRMS workstation interface to the RADC Automated Measurement System Worksheet E for automated source code reliability measurement.

Tables 3.8 through 3.11 provide an estimate of the effort expended to process each RPFOM worksheet for both software projects utilized in the study. Tables 3.8, 3.9 and 3.10 show the separate data collection and data entry effort for each worksheet at the System, CSCI and Unit levels, respectively. Estimates for the system level worksheets are approximations. The remaining estimates are based on timing a few samples of each worksheet and then averaging. Similarities on the average times for several of the CSCI level worksheets in Table 3.9 are due to the unavailability of detail design specifications and the generic answering of "no" to applicable questions.

In Tables 3.8, 3.9 and 3.10, the total number of CSCIs and Units are shown for each software project, and average worksheet time is accumulated accordingly. In the right-hand columns of these tables, this information is averaged and accumulated for both projects in combination. Table 3.11 summarizes all of this information into accumulated effort for data collection and data entry for all worksheets for the individual and combined projects.

Based on the personnel and software projects utilized, we believe these effort tables to be useful predictors of future software reliability prediction efforts for similar applications. SAIC and RTI data collection personnel qualifications were quite representative of those called for in Section 3.2.3. Sim/Stim is representative of a medium software development project, having a total 72K LOC and an average of 154 LOC per unit. SCENE is representative of a small project, having a total of 24K LOC and a very wide range of unit sizes which average 142 LOC each. Combined, Sim/Stim and SCENE represent small-to-medium project whose units average about 148 LOC. For larger projects, one can extrapolate effort estimates based on the information contained herein. A table is provided for this purpose in Section 3.7.

3.6.2 RPFOM Numbers

RPFOMs were computed both for the two test projects and for the unit and CSC integration test samples selected from these projects. The complete set of computations specified in Section 3.3.2 were performed for all four Software Development Life Cycle phases on each software project. This yielded the Application (A) and Development Environment (A*D) RPFOMs at the System level, and the refined Requirements and Design (A*D*S1) and Implementation (A*D*S1*S2) RPFOMs at the CSCI level. The Implementation RPFOM also was computed for each individual test sample. All RPFOMs and their underlying metrics are documented in Tables 3.13 through 3.17. Table 3.12 provides an index of their organization and content. Table 3.18 contains a summary of the test sample RPFOMs.

3.7 Findings and Conclusions

Analysis of the computed RPFOM values is documented in Chapter 5. This section presents conclusions regarding the RADC software reliability prediction methodology, based on the technical activities which were performed for this task and the effort expended. In addition, recommendations are made for continued

Table 3.8. System Level Worksheet Effort (Minutes)

Work Sheet	Activity	SIM/STIM			SCENE			COMBINED		
		#SYS	TIME		#SYS	TIME		#SYS	AVG	CUM
0	DC DE	1	10.0 0.5		1	10.0 0.5		2	10.0 0.5	20.0 1.0
1A	DC DE		60.0 0.25			60.0 0.25			60.0 0.25	120.0 0.5
1B	DC DE		60.0 1.0			60.0 1.0			60.0 1.0	120.0 2.0
Total	DC DE	1	130.0 1.75		1	130.0 1.75		2		260.0 3.5

Legend: DC - Data Collection
DE - Data Entry

Table 3.9. CSCI Level Worksheet Effort (Minutes)




Work Sheet	Activity	SIM/STIM			SCENE			COMBINED		
		#C's	AVG	CUM	#C's	AVG	CUM	#C's	AVG	CUM
2A	DC	4	0.5	2.0	1	0.5	0.5	5	0.5	2.5
	DE		1.0	4.0		1.0	1.0		1.0	5.0
2B	DC		0.5	2.0		0.5	0.5		0.5	2.5
	DE		1.0	4.0		1.0	1.0		1.0	5.0
2D	DC		1.0	4.0		1.0	1.0		1.0	5.0
	DE		1.0	4.0		1.0	1.0		1.0	5.0
3A	DC		0.5	2.0		0.5	0.5		0.5	2.5
	DE		0.5	2.0		0.5	0.5		0.5	2.5
3B	DC		0.5	2.0		0.5	0.5		0.5	2.5
	DE		0.5	2.0		0.5	0.5		0.5	2.5
3C	DC		1.0	4.0		1.0	1.0		1.0	5.0
	DE		0.5	2.0		0.5	0.5		0.5	2.5
4B	DC		40.0	160.0		40.0	40.0		40.0	200.0
	DE		1.0	4.0		1.0	1.0		1.0	5.0
10A	DC		0.5	2.0		0.5	0.5		0.5	2.5
	DE		1.0	4.0		1.0	1.0		1.0	5.0
10B	DC		0.5	2.0		0.5	0.5		0.5	2.5
	DE		1.0	4.0		1.0	1.0		1.0	5.0
10D	DC		0.5	2.0		0.5	0.5		0.5	2.5
	DE		1.0	4.0		1.0	1.0		1.0	5.0
11B	DC		120.0	480.0		120.0	120.0		120.0	600.0
	DE		3.0	12.0		3.0	3.0		3.0	15.0
Total Manhours	DC DE	4		11.0 0.8	1		2.8 0.2	5		13.8 1.0

Table 3.10. Unit Level Worksheet Effort (Minutes)

Work Sheet	Activity	SIM/STIM			SCENE			COMBINED		
		#U's	AVG	CUM	#U's	AVG	CUM	#U's	AVG	CUM
2C	DC	466			38*	4.75	181	620		
	DC		0.25	117	116	0.25	29		0.53	329
	DE		0.5	233	38*	1.5	57		0.56	347
	DE				116	0.5	58			
4A	DC		10.0	4660	154	10.0	1540		10.0	6200
	DE		2.0	932		2.0	308		2.0	1240
10C	DC				38*	13.0	494			
	DC		0.5	233	116	0.5	58		1.3	806
	DE		0.5	233	38*	0.5	19		0.5	310
	DE				116	0.5	58			
11A	DC		3.0	13980	154	3.0	4260		3.0	18600
	DE		0.5	2330		0.5	770		0.5	3100
Total Mandays	DC	466		39.6	154		14.4	620		54.0
	DE			7.8			2.7			10.5

* Documentation existed for these units only

Table 3.11. Worksheet Effort Summary by Level (Manweeks)

Work Sheet	Activity	SIM/STIM			SCENE			COMBINED		
		#	AVG	CUM	#	AVG	CUM	#	AVG	CUM
System	DC	1		0.27	1		0.27	2		0.54
	DE			0.00			0.00			0.01
CSCI	DC	4		0.28	1		0.07	5		0.35
	DE			0.02			0.01			0.03
Unit	DC	466		7.92	154		2.88	620		10.80
	DE			1.56			0.54			2.10
Total Manweeks	DC			8.47			3.22			11.69
	DE			1.58			0.55			2.14

research in the applied software reliability prediction technology specified in the SRPEG.

3.7.1 Support Tools and Database

The IRMS workstation (a Mac II-based system) was a very effective and low-cost mechanism for the management of large volumes of software project measures. Operationally, the RPFOM Database application has been easy to use and maintain, and has yielded high productivity. It is strongly recommended that the metrics which were not automated and the manual interface to AMS Worksheet E be automated in future versions of the IRMS in order to increase this productivity.

Developmentally, AMS and the DBMS were difficult to learn to use and were unpredictable in some aspects of their operation. Both were acquired in their first release versions, and to some extent we were field testing them. Documentation was minimal and no formal user training was available for either product. We worked through these difficulties and our subsequent operational application ran smoothly.

AMS is a very useful and sophisticated measurement automation tool. Its Worksheet E should be enhanced to extract all of the software measures required by SRPEG Worksheet 11A. In addition, it would be potentially very useful to correlate the other AMS worksheets with the SRPEG worksheets, and automate their interfaces. This would provide the mechanism to fully integrate the IRMS workstation developed under the current contract into the Software Life Cycle Support Environment (SLCSE) facility at RADC.

A number of faults were encountered with the AMS and they were reported.

It is expected that the fault in AMS which prevents it from reliably preprocessing FORTRAN INCLUDE files will be corrected in a subsequent revision. Meanwhile, it is essential that these statements be removed (or commented out) in source files when running Worksheet E. Further, our experience with this function of AMS shows it to be both expedient and economical to run AMS on large mainframe systems. Operation on a multi-user VMS VAX 780 or single-user MicroVAX 2000 was unacceptably slow. On the VAX 8650 the production rates were very acceptable for this heavily compute-bound source language processor.

The present study has demonstrated the utility of a Macintosh-based DBMS for software reliability prediction data management and analysis, and the potential for a Mac interface to VAX-resident software reliability tools. This interface could be realized by completing an automated connection between IRMS and AMS, and by incorporating IRMS enhancements to automate all metrics calculations.

3.7.2 Software Projects and Materials

The total available documentation and source code for Sim/Sum and SCENE includes the specially prepared detail design specifications (by project consultants) for each of the experiment test samples. These software development project materials adequately support the two primary objectives of the current task:

- a. To evaluate the software reliability prediction methodology in the SRPEG by utilizing the specially derived instructions for data collectors on complete software systems.

Table 3.12 RPFOM Number Index

Type	Level	Project	Table	Page
Project	CSCI	Sim/Stim	5-7	5-9
		SCENE	5-8	5-13
Samples	Integ	Sim/Stim	5-9	5-14
	Unit	Sim/Stim	5-10	5-18
		SCENE	5-11	5-22

Table 3.13. SIM/STIM Project RPFOMs

CSCI: PATEST - EXEC

PROJECT: AFAIDS SIM/STIM

	A	D ₁ or D ₂	S1		S2		
Phase	Metrics	Application (A)	Developmental Environment (D ₁) or (D ₂)	Anomaly Mgmt (SA) • (ST) • (SQ)	Traceability Review (SQ)	Language Modularity Complexity Review (SM) • (SX) • (SR) • (SR)	RPFOM Calculation
SRR	A	0.0123					0.012
	A • D ₁	0.0123	1.3				0.015
	A • D ₂	0.0123	2.0				0.024
SSR	A • D ₁ • S1 ₁	0.0123	1.3	1.1	1.1		0.021
	A • D ₂ • S1 ₁	0.0123	2.0	1.1	1.1		0.032
PDR	A • D ₁ • S1 ₂	0.0123	1.3	1.1	1.1		0.021
	A • D ₂ • S1 ₂	0.0123	2.0	1.1	1.1		0.032
ODR	A • D ₁ • S1 ₃	0.0123	1.3	1.1	1.1		0.021
	A • D ₂ • S1 ₃	0.0123	2.0	1.1	1.1		0.032
C&UT	A • D ₁ • S1 ₃ • S2	0.0123	1.3	1.1	1.1	1.0 0.928 1.040 1.5	0.030
	A • D ₂ • S1 ₃ • S2	0.0123	2.0	1.1	1.1	1.0 0.928 1.040 1.5	0.047

Table 3.13. SIM/STIM Project RPFOMs

Page: 2CSCI: PAMAINPROJECT: AFAIDS SIM/STIM

	A	D ₁ or D ₂	S1		S2		
Phase	Metrics	Application (A)	Developmental Environment (D ₁) or (D ₂)	Anomaly Mgmt (SA) • (ST) • (SQ)	Traceability Review (SR)	Language Modularity Complexity Review (SM) • (SX) • (SR) • (SR)	RPFOM Calculation
SRR	A	0.0123					0.012
	A • D ₁	0.0123	1.3				0.015
	A • D ₂	0.0123	2.0				0.024
SSR	A • D ₁ • S1 ₁	0.0123	1.3	1.1	1.1		0.021
	A • D ₂ • S1 ₁	0.0123	2.0	1.1	1.1		0.032
PDR	A • D ₁ • S1 ₂	0.0123	1.3	1.1	1.1		0.021
	A • D ₂ • S1 ₂	0.0123	2.0	1.1	1.1		0.032
CDR	A • D ₁ • S1 ₃	0.0123	1.3	1.1	1.1		0.021
	A • D ₂ • S1 ₃	0.0123	2.0	1.1	1.1		0.032
C&UT	A • D ₁ • S1 ₃ • S2	0.0123	1.3	1.1	1.1	1.0 0.916 1.033 1.5	0.030
	A • D ₂ • S1 ₃ • S2	0.0123	2.0	1.1	1.1	1.0 0.916 1.033 1.5	0.047

Table 3.13. SIM/STIM Project RPFOMs

Page: 3

CSCI: PAPRE-IESI
PROJECT: AFAIDS

Phase	Metrics	A	D ₁ or D ₂	S1		S2		RPFOM Calculation
		Application (A)	Developmental Environment (D ₁) or (D ₂)	Anomaly Mgmt (SA) • (ST) • (SQ)	Traceability Review (SQ)	Language Modularity Complexity Review (SM) • (SX) • (SR) • (SR)		
SRR	A	0.0123						0.0123
	A • D ₁	0.0123	1.3					0.0159
	A • D ₂	0.0123	2.0					0.0246
SSR	A • D ₁ • S1 ₁	0.0123	1.3	1.1	1.1			0.0210
	A • D ₂ • S1 ₁	0.0123	2.0	1.1	1.1			0.0320
PDR	A • D ₁ • S1 ₂	0.0123	1.3	1.1	1.1			0.0210
	A • D ₂ • S1 ₂	0.0123	2.0	1.1	1.1			0.0320
CDR	A • D ₁ • S1 ₃	0.0123	1.3	1.1	1.1			0.0210
	A • D ₂ • S1 ₃	0.0123	2.0	1.1	1.1			0.0320
C&UT	A • D ₁ • S1 ₃ • S2	0.0123	1.3	1.1	1.1	1.0	1.083 1.074 1.5	0.0370
	A • D ₂ • S1 ₃ • S2	0.0123	2.0	1.1	1.1	1.0	1.083 1.074 1.5	0.0570

Table 3.13. SIM/STIM Project RPFOMs

CSCI: PAPOST-TEST
PROJECT: AFAIDS

		A	D1 or D2	S1		S2		
Phase	Metrics	Application (A)	Developmental Environment (D1) or (D2)	Anomaly Mgmt (SA)	Traceability (ST)	Quality Review (SQ)	Language Modularity Complexity Review (SM) • (SX) • (SR) • (SR)	RPFOM Calculation
SRR	A	0.0123						0.012
	A • D1	0.0123	1.3					0.015
	A • D2	0.0123	2.0					0.024
SSR	A • D1 • S11	0.0123	1.3	1.1	1.1	1.1		0.021
	A • D2 • S11	0.0123	2.0	1.1	1.1	1.1		0.032
PDR	A • D1 • S12	0.0123	1.3	1.1	1.1	1.1		0.021
	A • D2 • S12	0.0123	2.0	1.1	1.1	1.1		0.032
QDR	A • D1 • S13	0.0123	1.3	1.1	1.1	1.1		0.021
	A • D2 • S13	0.0123	2.0	1.1	1.1	1.1		0.032
C&UT	A • D1 • S13 • S2	0.0123	1.3	1.1	1.1	1.1	1.0 0.946 1.315 1.5	0.029
	A • D2 • S13 • S2	0.0123	2.0	1.1	1.1	1.1	1.0 0.946 1.315 1.5	0.060

Table 3.14. SCENE Project RPFOMs

Page: 1

CSCI: PSSCENE

PROJECT: SCENE

Phase	Metrics	A	D ₁ or D ₂	S1			S2			RPFOM Calculation
		Application (A)	Developmental Environment (D ₁) or (D ₂)	Anomaly Mgmt (SA) • (ST)	Traceability (SQ)	Quality Review (SQ)	Language Modularity (SM) • (SX)	Complexity (SR)	Standards Review (SR)	
SRR	A	0.0123								0.012
	A • D ₁	0.0123	0.76							0.009
	A • D ₂	0.0123	0.5							0.006
SSR	A • D ₁ • S1 ₁	0.0123	0.76	1.1	1.1	1.1				0.012
	A • D ₂ • S1 ₁	0.0123	0.5	1.1	1.1	1.1				0.008
PDR	A • D ₁ • S1 ₂	0.0123	0.76	1.1	1.1	1.1				0.012
	A • D ₂ • S1 ₂	0.0123	0.5	1.1	1.1	1.1				0.008
CDR	A • D ₁ • S1 ₃	0.0123	0.76	1.1	1.1	1.1				0.012
	A • D ₂ • S1 ₃	0.0123	0.5	1.1	1.1	1.1				0.008
C&UT	A • D ₁ • S1 ₃ • S2	0.0123	0.76	1.1	1.1	1.1	1.0	0.989	1.067	1.0
	A • D ₂ • S1 ₃ • S2	0.0123	0.5	1.1	1.1	1.1	1.0	0.989	1.067	1.0
										0.013
										0.008

Table 3.15. SIM/STIM Integration Sample RPFOMs

Page: 1

CSCI: PASNWMSN1

PROJECT: AFAIDS

Phase	Metrics	A	D ₁ or D ₂	S1		S2		RPFOM Calculation
				Anomaly Mgmt (SA) •	Traceability (ST) •	Quality Review (SQ)	Language Modularity Complexity Review (SM) • (SX) • (SR) •	
SRR	A	0.0123						0.012
	A • D ₁	0.0123	1.3					0.015
	A • D ₂	0.0123	2.0					0.024
SSR	A • D ₁ • S1 ₁	0.0123	1.3	1.1	1.1	1.1		0.021
	A • D ₂ • S1 ₁	0.0123	2.0	1.1	1.1	1.1		0.032
PDR	A • D ₁ • S1 ₂	0.0123	1.3	1.1	1.1	1.1		0.021
	A • D ₂ • S1 ₂	0.0123	2.0	1.1	1.1	1.1		0.032
CDR	A • D ₁ • S1 ₃	0.0123	1.3	1.1	1.1	1.1		0.021
	A • D ₂ • S1 ₃	0.0123	2.0	1.1	1.1	1.1		0.032
CAUT	A • D ₁ • S1 ₃ • S2	0.0123	1.3	1.1	1.1	1.1	1.0 0.9 1.1	0.021
	A • D ₂ • S1 ₃ • S2	0.0123	2.0	1.1	1.1	1.1	1.0 0.9 1.1	0.032

Table 3.15. SIM/STIM Integration Sample RPFOMs

Page: 2

CSCI: PATHURDI
PROJECT: AFAIDS

Phase	Metrics	A	D ₁ or D ₂	S1		S2		RPFOM Calculation
		Application (A)	Developmental Environment (D ₁) or (D ₂)	Anomaly Mgmt (SA) • (ST) • (SQ)	Tracability Review (SQ)	Language Modularity Complexity Review (SM) • (SX) • (SR) • (SR)	Standards Review (SR)	
SRR	A	0.0123						0.012
	A • D ₁	0.0123	1.3					0.015
	A • D ₂	0.0123	2.0					0.024
SSR	A • D ₁ • S1 ₁	0.0123	1.3	1.1	1.1			0.021
	A • D ₂ • S1 ₁	0.0123	2.0	1.1	1.1			0.032
PDR	A • D ₁ • S1 ₂	0.0123	1.3	1.1	1.1			0.021
	A • D ₂ • S1 ₂	0.0123	2.0	1.1	1.1			0.032
CDR	A • D ₁ • S1 ₃	0.0123	1.3	1.1	1.1			0.021
	A • D ₂ • S1 ₃	0.0123	2.0	1.1	1.1			0.032
C&UT	A • D ₁ • S1 ₃ • S2	0.0123	1.3	1.1	1.1	1.0	1.333 0.866 1.0	0.024
	A • D ₂ • S1 ₃ • S2	0.0123	2.0	1.1	1.1	1.0	1.333 0.866 1.0	0.037

Table 3.15. SIM/STIM Integration Sample RPFOMS

Page: 3

CSCI: PATHUADSPROJECT: AFAIDS

	A	D ₁ or D ₂	S1		S2				
Phase	Metrics	Application (A)	Developmental Environment (D ₁) or (D ₂)	Acromoly Mgmt (SA) • Traceability (ST) • Quality Review (SQ)	Language Modularity Complexity Standards Review (SR) • (SR) • (SR) • (SR)				RPFOM Calculation
SRR	A	0.0123							0.012
	A • D ₁	0.0123	1.3						0.015
	A • D ₂	0.0123	2.0						0.024
SSR	A • D ₁ • S1 ₁	0.0123	1.3	1.1 1.1 1.1	1.1	1.1	1.1		0.021
	A • D ₂ • S1 ₁	0.0123	2.0	1.1 1.1 1.1	1.1	1.1	1.1		0.032
PDR	A • D ₁ • S1 ₂	0.0123	1.3	1.1 1.1 1.1	1.1	1.1	1.1		0.021
	A • D ₂ • S1 ₂	0.0123	2.0	1.1 1.1 1.1	1.1	1.1	1.1		0.032
CDR	A • D ₁ • S1 ₃	0.0123	1.3	1.1 1.1 1.1	1.1	1.1	1.1		0.021
	A • D ₂ • S1 ₃	0.0123	2.0	1.1 1.1 1.1	1.1	1.1	1.1	•	0.032
C&UT	A • D ₁ • S1 ₃ • S2	0.0123	1.3	1.1 1.1 1.1	1.1	1.1	1.1	1.0 1.5 0.9 1.0	0.028
	A • D ₂ • S1 ₃ • S2	0.0123	2.0	1.1 1.1 1.1	1.1	1.1	1.1	1.0 1.5 0.9 1.0	0.044

Table 3.15. SIM/STIM Integration Sample RPFOMs

Page: 4

CSCI: PATHPCONPROJECT: AFAIDS

	A	D ₁ or D ₂	S1		S2		RPFOM Calculation
Phase	Metrics	Application (A)	Developmental Environment (D ₁) or (D ₂)	Anomaly Mgmt (SA) • Traceability (ST) • Quality Review (SQ)	Language Modularity Complexity Review (SM) • (SX) • (SR) • (SF)		
SPR	A	0.0123					0.012
	A • D ₁	0.0123	1.3				0.015
	A • D ₂	0.0123	2.0				0.024
SSR	A • D ₁ • S11	0.0123	1.3	1.1 1.1 1.1			0.021
	A • D ₂ • S11	0.0123	2.0	1.1 1.1 1.1			0.032
PDR	A • D ₁ • S12	0.0123	1.3	1.1 1.1 1.1			0.021
	A • D ₂ • S12	0.0123	2.0	1.1 1.1 1.1			0.032
CDR	A • D ₁ • S13	0.0123	1.3	1.1 1.1 1.1			0.021
	A • D ₂ • S13	0.0123	2.0	1.1 1.1 1.1			0.032
C&UT	A • D ₁ • S13 • S2	0.0123	1.3	1.1 1.1 1.1	1.0 1.0 0.933	1.0	0.019
	A • D ₂ • S13 • S2	0.0123	2.0	1.1 1.1 1.1	1.0 1.0 0.933	1.0	0.030

Table 3.16. SIM/STIM Unit Sample RPFOMs

CSCI: PASNWMSN2PROJECT: AFATDS

	A	D ₁ or D ₂	S1			S2			RPFOM Calculation
Phase	Metrics	Application (A)	Developmental Environment (D ₁) or (D ₂)	Anomaly Mgmt (SA)	Traceability (ST)	Quality Review (SQ)	Language Modularity Complexity (SM) • (SX)	Standards Review (SR)	
SRR	A	0.0123							0.012
	A • D ₁	0.0123	1.3						0.015
	A • D ₂	0.0123	2.0						0.024
SSH	A • D ₁ • S1 ₁	0.0123	1.3	1.1	1.1	1.1			0.021
	A • D ₂ • S1 ₁	0.0123	2.0	1.1	1.1	1.1			0.032
PDR	A • D ₁ • S1 ₂	0.0123	1.3	1.1	1.1	1.1			0.021
	A • D ₂ • S1 ₂	0.0123	2.0	1.1	1.1	1.1			0.032
ODR	A • D ₁ • S1 ₃	0.0123	1.3	1.1	1.1	1.1			0.021
	A • D ₂ • S1 ₃	0.0122	2.0	1.1	1.1	1.1			0.032
C&UT	A • D ₁ • S1 ₃ • S2	0.0123	1.3	1.1	1.1	1.1	1.0 0.9 1.5	0.75	0.021
	A • D ₂ • S1 ₃ • S2	0.0123	2.0	1.1	1.1	1.1	1.0 0.9 1.5	0.75	0.033

CSCI: PATHXBDI
PROJECT: AFAIDS

Table 3.16. SIM/STIM Unit Sample RPFOMs

Page: 2

	A	D ₁ or D ₂	S1		S2		RPFOM Calculation
Phase	Metrics	Application (A)	Developmental Environment (D ₁) or (D ₂)	Anomaly Mgmt (SA) • Traceability (ST) • (ST) • (SQ)	Quality Review (SQ)	Language Modularity Complexity Review (SM) • (SX) • (SR) • (SR)	
SRR	A	0.0123					0.012
	A • D ₁	0.0123	1.3				0.015
	A • D ₂	0.0123	2.0				0.024
SSR	A • D ₁ • S1 ₁	0.0123	1.3	1.1	1.1		0.021
	A • D ₂ • S1 ₁	0.0123	2.0	1.1	1.1		0.032
PDR	A • D ₁ • S1 ₂	0.0123	1.3	1.1	1.1		0.021
	A • D ₂ • S1 ₂	0.0123	2.0	1.1	1.1		0.032
CDR	A • D ₁ • S1 ₃	0.0123	1.3	1.1	1.1		0.021
	A • D ₂ • S1 ₃	0.0123	2.0	1.1	1.1		0.032
C&UT	A • D ₁ • S1 ₃ • S2	0.0123	1.3	1.1	1.1	1.0 2.0 1.0 1.0	0.042
	A • D ₂ • S1 ₃ • S2	0.0123	2.0	1.1	1.1	1.0 2.0 1.0 1.0	0.065

Table 3.16. SIM/STIM Unit Sample RPFOMs

Page: 3

CSCI: PATHUADCPROJECT: AFAIDS

	A	D ₁ or D ₂	S1		S2			RPFOM Calculation
Phase	Metrics	Application (A)	Developmental Environment (D ₁) or (D ₂)	Anomaly Mgmt (SA) • Traceability (ST) • (SQ)	Quality Review (SR)	Language Modularity (SM) • (SX) • (SR) • (SR)	Standards Complexity Review (SR)	
SFR	A	0.0123						0.012
	A • D ₁	0.0123	1.3					0.015
	A • D ₂	0.0123	2.0					0.024
SSR	A • D ₁ • S1 ₁	0.0123	1.3	1.1 1.1 1.1	1.1			0.021
	A • D ₂ • S1 ₁	0.0123	2.0	1.1 1.1 1.1	1.1			0.032
PDR	A • D ₁ • S1 ₂	0.0123	1.3	1.1 1.1 1.1	1.1			0.021
	A • D ₂ • S1 ₂	0.0123	2.0	1.1 1.1 1.1	1.1			0.032
QDR	A • D ₁ • S1 ₃	0.0123	1.3	1.1 1.1 1.1	1.1			0.021
	A • D ₂ • S1 ₃	0.0123	2.0	1.1 1.1 1.1	1.1			0.032
C&UT	A • D ₁ • S1 ₃ • S2	0.0123	1.3	1.1 1.1 1.1	1.1	1.0 2.0 1.0	1.0	0.042
	A • D ₂ • S1 ₃ • S2	0.0123	2.0	1.1 1.1 1.1	1.1	1.0 2.0 1.0	1.0	0.065

Table 3.16. SIM/STIM Unit Sample RPFOMs

Page: 4

CSCI: PATHUSCN
PROJECT: AFAIDS

Phase	Metrics	A	D ₁ or D ₂	S1			S2			RPFOM Calculation
		Application (A)	Developmental Environment (D ₁) or (D ₂)	Anomaly Mgmt (SA)	Traceability (ST)	Quality Review (SQ)	Language Modularity (SM)	Complexity (SX)	Standards Review (SR)	
SPR	A	0.0123								0.012
	A • D ₁	0.0123	1.3							0.015
	A • D ₂	0.0123	2.0							0.024
SSR	A • D ₁ • S1 ₁	0.0123	1.3	1.1	1.1	1.1				0.021
	A • D ₂ • S1 ₁	0.0123	2.0	1.1	1.1	1.1				0.032
PDR	A • D ₁ • S1 ₂	0.0123	1.3	1.1	1.1	1.1				0.021
	A • D ₂ • S1 ₂	0.0123	2.0	1.1	1.1	1.1				0.032
QDR	A • D ₁ • S1 ₃	0.0123	1.3	1.1	1.1	1.1				0.021
	A • D ₂ • S1 ₃	0.0123	2.0	1.1	1.1	1.1				0.032
C&UT	A • D ₁ • S1 ₃ • S2	0.0123	1.3	1.1	1.1	1.1	1.0	1.0	1.0	0.021
	A • D ₂ • S1 ₃ • S2	0.0123	2.0	1.1	1.1	1.1	1.0	1.0	1.0	0.032

Table 3.17. SCENE Unit Sample RPFOMs

Page: 1

CSCI: PSSCANNER

PROJECT: SCENE

Phase	Metrics	A	D ₁ or D ₂	S1		S2			RPFOM Calculation
				Anomaly Mgmt (SA) •	Traceability (ST) •	Quality Review (SQ)	Language Modularity (SM) •	Complexity (SR) •	
SRR	A	0.0123							0.012
	A • D ₁	0.0123	0.76						0.009
	A • D ₂	0.0123	0.5						0.006
SSR	A • D ₁ • S1 ₁	0.0123	0.76	1.1	1.1	1.1			0.012
	A • D ₂ • S1 ₁	0.0123	0.5	1.1	1.1	1.1			0.008
PDR	A • D ₁ • S1 ₂	0.0123	0.76	1.1	1.1	1.1			0.012
	A • D ₂ • S1 ₂	0.0123	0.5	1.1	1.1	1.1			0.008
ODR	A • D ₁ • S1 ₃	0.0123	0.76	1.1	1.1	1.1			0.012
	A • D ₂ • S1 ₃	0.0123	0.5	1.1	1.1	1.1			0.008
C&UT	A • D ₁ • S1 ₃ • S2	0.0123	0.76	1.1	1.1	1.1	1.0	1.0	0.018
	A • D ₂ • S1 ₃ • S2	0.0123	0.5	1.1	1.1	1.1	1.0	1.0	0.012

Table 3.17. SCENE Unit Sample RPFOMs

Page: 2

CSCI: PSINFANSENPROJECT: SCENE

		A	D ₁ or D ₂	S1	S2	
Phase	Metrics	Application (A)	Developmental Environment (D ₁) or (D ₂)	Anomaly Mgmt (SA) • (ST) • (SQ)	Language Modularity Complexity Review (SM) • (SX) • (SR) • (SR)	RPFOM Calculation
SRR	A	0.0123				0.012
	A • D ₁	0.0123	0.76			0.009
	A • D ₂	0.0123	0.5			0.006
SSR	A • D ₁ • S1 ₁	0.0123	0.76	1.1 1.1 1.1		0.012
	A • D ₂ • S1 ₁	0.0123	0.5	1.1 1.1 1.1		0.008
FDR	A • D ₁ • S1 ₂	0.0123	0.76	1.1 1.1 1.1		0.012
	A • D ₂ • S1 ₂	0.0123	0.5	1.1 1.1 1.1		0.008
ODR	A • D ₁ • S1 ₃	0.0123	0.76	1.1 1.1 1.1		0.012
	A • D ₂ • S1 ₃	0.0123	0.5	1.1 1.1 1.1		0.008
C&UT	A • D ₁ • S1 ₃ • S2	0.0123	0.76	1.1 1.1 1.1	1.0 1.0 1.5 1.5	0.027
	A • D ₂ • S1 ₃ • S2	0.0123	0.5	1.1 1.1 1.1	1.0 1.0 1.5 1.5	0.018

CSCI: PSINPSAT
PROJECT: SCENE

Table 3.17. SCENE Unit Sample RPFOMs

Page: 3

		A	D ₁ or D ₂	S1		S2			RPFOM Calculation
Phase	Metrics	Application (A)	Developmental Environment (D ₁) or (D ₂)	Anomaly Mgmt (SA) • (ST)	Traceability (SQ)	Quality Review (SR)	Language Modularity (SM) • (SX)	Complexity (SR) • (SR)	
SRR	A	0.0123							0.012
	A • D ₁	0.0123	0.76						0.009
	A • D ₂	0.0123	0.5						0.006
SSR	A • D ₁ • S1 ₁	0.0123	0.76	1.1	1.1	1.1			0.012
	A • D ₂ • S1 ₁	0.0123	0.5	1.1	1.1	1.1			0.008
PDR	A • D ₁ • S1 ₂	0.0123	0.76	1.1	1.1	1.1			0.012
	A • D ₂ • S1 ₂	0.0123	0.5	1.1	1.1	1.1			0.008
CDR	A • D ₁ • S1 ₃	0.0123	0.76	1.1	1.1	1.1			0.012
	A • D ₂ • S1 ₃	0.0123	0.5	1.1	1.1	1.1			0.008
C&UT	A • D ₁ • S1 ₃ • S2	0.0123	0.76	1.1	1.1	1.1	1.0	2.0	1.5
	A • D ₂ • S1 ₃ • S2	0.0123	0.5	1.1	1.1	1.1	1.0	2.0	1.5
									0.055
									0.036

Table 3.17. SCENE Unit Sample RPFOMS

Page: 4

CSCI: PSINMODEPROJECT: SCENE

		A	D ₁ or D ₂	S1		S2			RPFOM Calculation
Phase	Metrics	Application (A)	Developmental Environment (D ₁) or (D ₂)	Anomaly Mgmt (SA) • (ST) • (SQ)	Quality Review (SQ)	Language Modularity Complexity (SM) • (SX) • (SR)	Standards Review (SR)		
SPR	A	0.0123						0.012	
	A • D ₁	0.0123	0.76					0.009	
	A • D ₂	0.0123	0.5					0.006	
SSR	A • D ₁ • S1 ₁	0.0123	0.76	1.1	1.1			0.012	
	A • D ₂ • S1 ₁	0.0123	0.5	1.1	1.1			0.008	
PDR	A • D ₁ • S1 ₂	0.0123	0.76	1.1	1.1			0.012	
	A • D ₂ • S1 ₂	0.0123	0.5	1.1	1.1			0.008	
CDR	A • D ₁ • S1 ₃	0.0123	0.76	1.1	1.1			0.012	
	A • D ₂ • S1 ₃	0.0123	0.5	1.1	1.1			0.008	
C&UT	A • D ₁ • S1 ₃ • S2	0.0123	0.76	1.1	1.1	1.0	1.0	1.5	0.027
	A • D ₂ • S1 ₃ • S2	0.0123	0.5	1.1	1.1	1.0	1.0	1.5	0.018

Table 3.18. Task III RPFOM Summary

Test Project	Test Level	Sample Name	RPFOM Components					RPFOM Number
			A	D1	D2	S1 ₃	S2	
Sim/Stim	Integ	SNWMSN	0.123	1.30	2.0	1.1	0.990	0.021
		THURDT	0.123	1.30	2.0	1.1	1.154	0.023
		THUADS	0.123	1.30	2.0	1.1	1.350	0.024
		THPCON	0.123	1.30	2.0	1.1	0.933	0.037
Sim/Stim	Unit	SNWMSN	0.123	1.30	2.0	1.1	1.013	0.028
		THXRDT	0.123	1.30	2.0	1.1	2.000	0.044
		THUADC	0.123	1.30	2.0	1.1	2.000	0.019
		THUSCN	0.123	1.30	2.0	1.1	1.000	0.030
SCENE	Unit	SCANNER	0.123	0.76	0.5	1.1	1.500	
		INFANSEN	0.123	0.76	0.5	1.1	2.250	
		INPSAT	0.123	0.76	0.5	1.1	4.500	
		INMODE	0.123	0.76	0.5	1.1	2.250	

- b. To provide RPFOMs for test samples selected from these systems for comparison with both the original development fault densities and experimental testing results.

For Sim/Stim the available PDL is not sufficiently detailed to collect the Anomaly Management (SA), Traceability (ST) and Quality Review (SQ) metrics. For SCENE, detail design specifications are not available for most of the units. We were unable to collect ST on any of them; SA and SQ could be collected on only 38 units. Still, this incomplete condition of the project documentation is meaningful in our study since such unavailability is factored directly into the RPFOM computations.

The SCENE detail design specifications are as-coded versions of the software. Thus, they are not appropriate to assess as "predictors" of that software's subsequent reliability during test and evaluation. They have been utilized in the current study exclusively to assess the SRPEG and RDCI RPFOM methodology and to provide data for future use in estimating software reliability prediction efforts.

The Sim/Stim and SCENE projects can be reused to compare and validate future refinements to the current RADC/SAIC software reliability prediction methodology. MC³ and NTC also can be utilized to produce additional data points on current and future findings and conclusions. But the most effective application of this methodology requires a closer adherence to DoD-STD-2167A than observed by projects available for this study (and, in effect, by most software development projects in general). The inverse is also true; i.e., a requirement for the RADC software reliability prediction methodology on software development projects would force closer adherence to 2167A. These requirements depend upon government policy and enforcement, and on sponsorship of additional studies in these areas in order to gain the desired cost benefits in the long run.

3.7.3 Data Collection and Calculation

Data collection and recording on the provided work sheets and answer sheets is primarily a technical support process. It is guided by the prepared instructions and can be scheduled in terms of activity sequences, resource and personnel requirements, and level-of-effort allocations. Essentially, all procedures can be performed by junior technical and administrative personnel, except for the questions on Worksheet 2C. These need to be answered by a programmer/QA analyst.

Data entry and RPFOM calculations utilizing the RPFOM Database are primarily administrative support processes. Also guided by the prepared instructions, they can be budgeted and scheduled similarly to data collection and recording.

Tables 3.8 through 3.11 show the estimated time spent on the various RPFOM work sheets and answer sheets for both projects utilized for this study. This information is summarized in Table 3-19. As you can see, most of the time was spent on worksheets 1A, 1B, 4B and 11B. Data collection, when compared to data entry, represented most of the total time. The average times for Worksheets 2A, 2B, 3A, 3B, 3C, 10A and 10B reflect the unavailability of requisite source documentation. These worksheets were completed by entering "no" answers directly into the database. Consequently the average time expended is considerably lower than if the answers had been extracted from available documents. The average times for Worksheets 11A and 11B are low since only two-thirds of the questions were answered using AMS. Future study can provide additional effort data for these particular worksheets.

Using the information in Tables 3.7 and 3.11, it is possible to estimate the level of effort which will be required to collect and compute RPFOMs for various sized projects. Table 3.20 provides this information. These projections are simply extrapolations of the level of effort expended in the current study: first by estimated worksheet times which are then factored by "1.8" in order to account for all associated planning, supervision, set-up and administrative time. This weighting derives from the 25 manweeks required to perform the complete task, of which 13.83 manweeks were spent specifically on data collection and data entry operations.

Table 3.20 Average RPFOM Effort by Work Sheet (Minutes)

Work Sheet	Activity	Average Time
0	DC	10.00
	DE	0.50
1A	DC	60.00
	DE	0.25
1B	DC	60.00
	DE	1.00
<hr/>		
2A	DC	0.50
	DE	1.00
2B	DC	0.50
	DE	1.00
2D	DC	1.00
	DE	1.00
3A	DC	0.50
	DE	0.50
3B	DC	0.50
	DE	0.50
3C	DC	1.00
	DE	0.50
4B	DC	40.00
	DE	1.00
10A	DC	0.50
	DE	1.00
10B	DC	0.50
	DE	1.00
10D	DC	0.50
	DE	1.00
11B	DC	120.00
	DE	3.00
<hr/>		
2C	DC	0.53
	DE	0.56
4A	DC	10.00
	DE	2.00
10C	DC	1.30
	DE	0.50
11A	DC	3.00
	DE	0.50
<hr/>		
Total Hours	DC	5.17
	DE	0.28

Table 3.21 Estimated RPFOM Effort for Projects (Manweeks)

Work Sheet	Level	Time Per W/S	Time Per Project (LOC)			
			25K	50K	100K	250K
0	System	10.50	0.01	0.01	0.01	0.01
1A		60.25	0.03	0.03	0.03	0.03
1B		61.0	0.03	0.03	0.03	0.03
2A	CSCI	1.50	0.01	0.01	0.01	0.01
2B		1.50	0.01	0.01	0.01	0.01
2D		2.00	0.01	0.01	0.01	0.01
3A		1.00	0.01	0.01	0.01	0.01
3B		1.00	0.01	0.01	0.01	0.01
3C		1.50	0.01	0.01	0.01	0.01
4B		4.00	0.03	0.05	0.09	0.23
10A		1.50	0.01	0.01	0.01	0.01
10B		1.50	0.01	0.01	0.01	0.01
10D		1.50	0.01	0.01	0.01	0.01
11B		123.00	0.07	0.14	0.27	0.68
2C	Unk	1.09	0.08	0.15	0.29	0.73
4A		12.00	0.81	1.62	3.23	8.08
10C		1.80	0.13	0.25	0.49	1.23
11A		3.50	2.35	4.70	9.40	23.50
Work Sheet Subtotal			3.63	6.94	13.93	34.54
Total (1.8)			6.53	12.49	25.07	62.17

4.0 SOFTWARE TESTING EXPERIMENT

To date, intuition and the advice of experts have guided much of the work done in the fields of software testing and software reliability measurement. Controlled software experiments require extensive capital commitment and a large level of effort and thus have rarely been performed.

This formal experiment in software testing and reliability is a major step in providing much needed data, obtained in a controlled environment. Section 4.5.5.3 describes the raw test results data which was collected from the application of six standard testing techniques at the unit and CSC integration test levels utilizing test samples from two AF/DoD software projects. Descriptive and statistical analyses of these raw results data are provided in Chapter 5.

4.1 Technical Approach

Software testing was conducted in accordance with the experiment design, as specified in the SRTIP. Figure 4.1 illustrates the software testing process. Due to time and effort constraints, the unit and CSC integration test samples identified in the SRTIP were replaced with test samples which could be easily removed from their software build environments and tested in a stand-alone manner with test drivers.

The principal elements of the software testing experiment which are described in this report are:

- a. The formal framework of the experiment, involving application of six commonly used software testing techniques at different test levels, utilizing code samples from previously developed software test projects.
- b. Conduct of the experiment
- c. Tester and testing technique performance measurement and software reliability estimation measurement.
- d. Collection, documentation and reporting of all tests, test conduct and test results based on the DoD 2167 standard and the goals of the SRTIP.

Test techniques to be evaluated were divided into two categories: deterministic techniques and nondeterministic techniques. Deterministic techniques are those for which one can determine without doubt that applying a given test technique to a given code sample will find a given error. (Note that while one *can* determine error without doubt, the potential for human error in the *determination* process still exists. Thus deterministic does not imply foolproof.) Nondeterministic techniques are those in which variability across such factors as test personnel and software characteristics can have profound effects on the effectiveness of the technique.

This distinction between test techniques is made to conserve time and resources. Techniques which can be highly and consistently automated and are less dependent upon tester expertise were seen as deterministic and suitable for empirical study. Nondeterministic techniques require experiments with several testers employing the test technique to the best of their ability on a given code sample. Experimental results obtained show whether applying the technique standardly and consistently

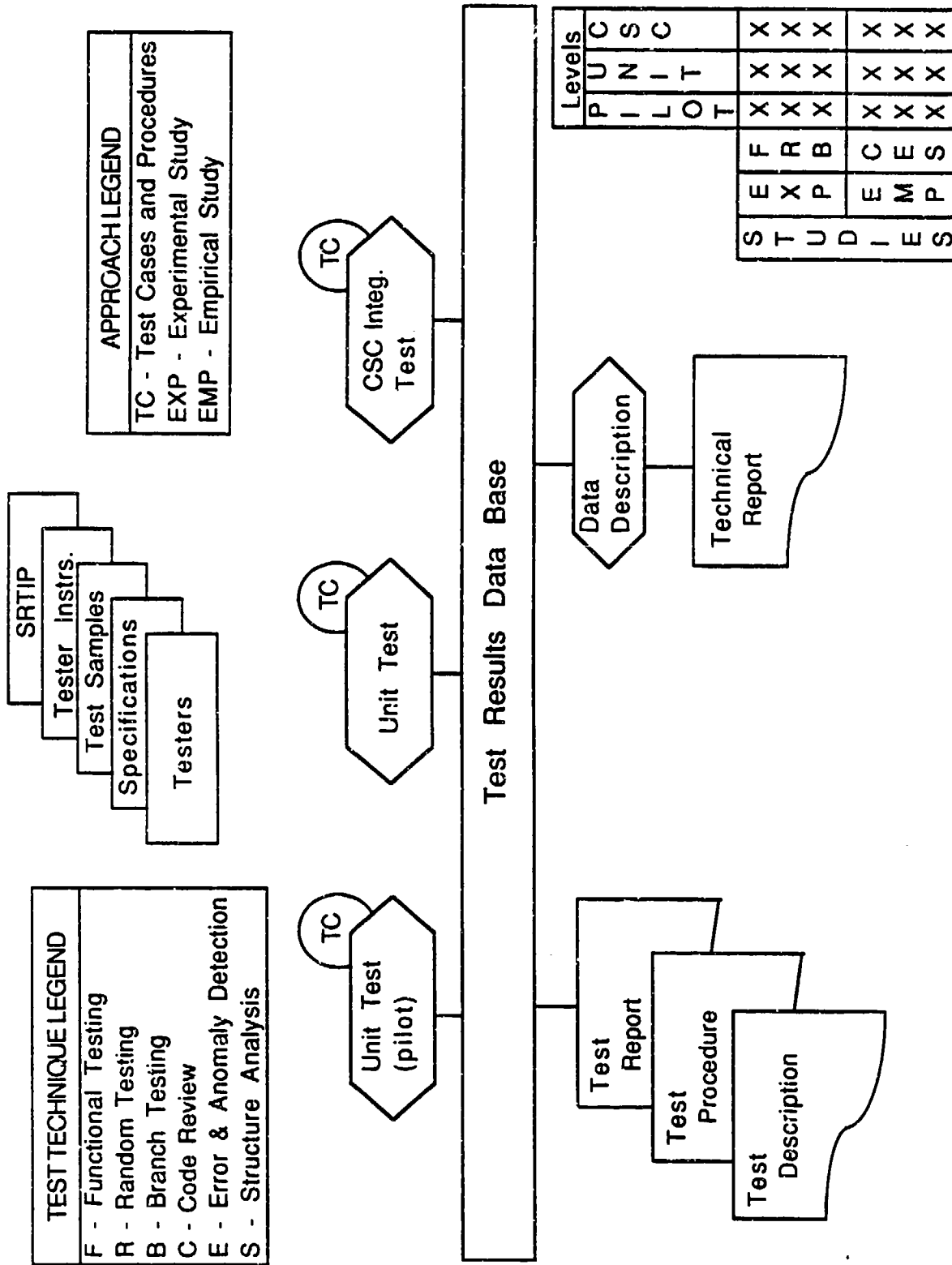


Figure 4.1 Software Testing Process

finds a given error. In this effort, four testers were employed to write test descriptions and test procedures for each test technique in the experiments.

A pilot study was performed prior to the complete set of experiments. This initial experimentation was conducted according to the developed design and protocol as a small, integral portion of the entire experiment. The experimental process and products were closely monitored during the pilot, and the knowledge gained was applied to improve the process for the remainder of the experimentation.

The experiments are designed as Latin Squares and involved four testers repetitively testing code samples with different test techniques. The variability in results due to the nondeterministic nature of the techniques can be averaged over all the testers and code samples. This increases confidence that what is measured is a test technique's performance, not a tester's performance.

Three of the experimental test techniques are dynamic, in that they involve executing the code and failures are discovered via testing. These techniques are random testing, functional testing, and branch testing. The fourth experimental test technique, code reading, is static, in that the code is not executed and faults are not discovered via testing. Both empirical study test techniques (structure analysis, and error and anomaly detection) are static techniques. In order to standardize the test recording process, faults and failures for both dynamic and static techniques were reported similarly, as errors, on Software Problem Reports (SPRs) during testing. The experiments spanned both the unit test and CSC integration test levels. They used samples from two Air Force/DoD software projects. CSCI level tests were not performed due to cost, time, and requirements for a CSCI/System level test configuration. Eight code samples, four from each of two projects, were used as experimental subjects at the unit test levels. Four code samples from one project were used at the CSC integration test level. Code samples were selected based on error proneness and accessibility to stand-alone testing with a driver. Table 4.1 shows the software projects and test levels for which code samples were utilized.

Table 4.1. Test Projects and Levels

Test Level	TEST TECHNIQUE					
	Experiment				Empirical Study	
	Functional Testing	Random Testing	Branch Testing	Code Review	Error & Anomaly	Structure Analysis
UNIT	Sim/Stim SCENE	Sim/Stim SCENE	Sim/Stim SCENE	Sim/Stim SCENE	Sim/Stim SCENE	Sim/Stim SCENE
CSC	Sim/Stim	Sim/Stim	Sim/Stim	Sim/Stim	Sim/Stim	Sim/Stim

The Latin Square framework, applied on a test level basis, organizes the experiments in a cohesive manner. Its benefits include the following:

- a. Efficient, continuous scheduling of testers and operators for the duration of the experiment conduct.
- b. Incremental execution, so that a pilot experiment can be performed first.
- c. Fairly straightforward method of analysis of the resulting data.

A detailed instruction manual was prepared utilizing information in the STH, SRPEG and SRTIP. These *Instructions for Testers and Operators* [11] provided a uniform standard for the application of the various testing techniques to each test sample by each tester. Special emphasis was placed on collecting the necessary test execution and error data to evaluate each test technique for its efficiency and effectiveness, test effort and test coverage, and to compare the techniques based upon these factors.

The experiment activities were performed jointly by Science Application International Corporation (SAIC) personnel in San Diego, CA, and Research Triangle Institute (RTI) personnel in Research Triangle Park, NC. As the prime contractor, SAIC was responsible for the overall experiment conduct and provided the larger level of effort. RTI, the principle subcontractor, supported each of the designated experiment setup, pilot experiment, and extended experiment activities, and performed an important advisory role.

4.2 Test Resources

The resources utilized to perform the experiment activities are described in this section. Included are computing systems, support software, software project code samples, personnel and tester instructions. All experiment activities were performed at the separate facilities of SAIC in San Diego, California and RTI in Research Triangle Park (RTP), North Carolina. Both facilities provided qualified technical project personnel, necessary project background/research data, and combined computation and documentation resources. There were no requirements or provisions for classified processing or security-related tests.

4.2.1 Computing System

The same DEC and Apple computer systems used during the data collection activities were also utilized at SAIC in San Diego and RTI in RTP in support of all experiment activities throughout the performance of this task. SAIC provided alternate VAX systems (a VAX 11/780, later replaced by a VAX 8550) to support software testing. These systems operated under the VAX/VMS version V4.7 operating system. RTI provided one VAX station 2000 running the VMS 4.7 operating system for the same use. Electronic communications between the SAIC and RTI sites was provided via ARPANET for VAX-to-VAX communications.

The IRMS, were a Macintosh II-based system, served as the automated workstation for performing test data management, calculations and report generation.

Figure 4.2 shows the software test configuration for experimental testing. The test project software samples and interfacing/support software programs were installed on the VAX/MicroVAX systems at both of the test sites. All of the collected test data was entered into the Mac II workstations where the descriptive analysis were

performed. Supplemental statistical analyses will be performed on the RTI MicroVAX 2000 utilizing SAS.

4.2.2 System Software

The following system software was utilized to conduct the software test experiment and empirical studies.

4.2.2.1 Communications.

ARPANET was used for VAX-to-VAX communications. Terminal emulation software (e.g., Macterminal and Mac 240) was used for Mac II-to-VAX communications at RTI.

4.2.2.2 Operating Systems.

VMS release 4.7 was used on the SAIC and RTI VAX systems.

4.2.2.3 Compiler.

VMS FORTRAN release V4.7 was used to compile all software code samples and all interfacing software developed by the testers.

4.2.2.4 Data Management and Analysis.

Three software tools were utilized to manage and evaluate all test data. These tools are shown in Table 4.2.

4.2.3 Test/Support Tools.

Three software testing and support tools were utilized to support each of the software testing techniques. These tools were used in the same capacity at both sites and are shown in Table 4.3.

4.2.3.1 DEC Test Manager

DTM [21] was used to organize online test information for the three dynamic testing techniques. Within the DTM, testers define test descriptions; each test description defines one test, by associating together the appropriate test sample, its ancillary data files (if any), the sample driver, input case(s), expected outputs (the benchmark), and actual test outputs. One or more test descriptions are organized into a DTM test collection. The DTM automatically stores test outputs and compares them with expected outputs (the benchmark), flagging all mismatches between actual and expected outputs.

4.2.3.2 SAIC SDDL

SAIC SDDL [23] was used in conjunction with the code review technique only. All code samples were run through the SDDL tool by support personnel prior to the beginning of this study. Each tester received the printed outputs of SDDLs static processing of their code samples. These outputs consist of:

- a. Table of contents for SDDLs output.

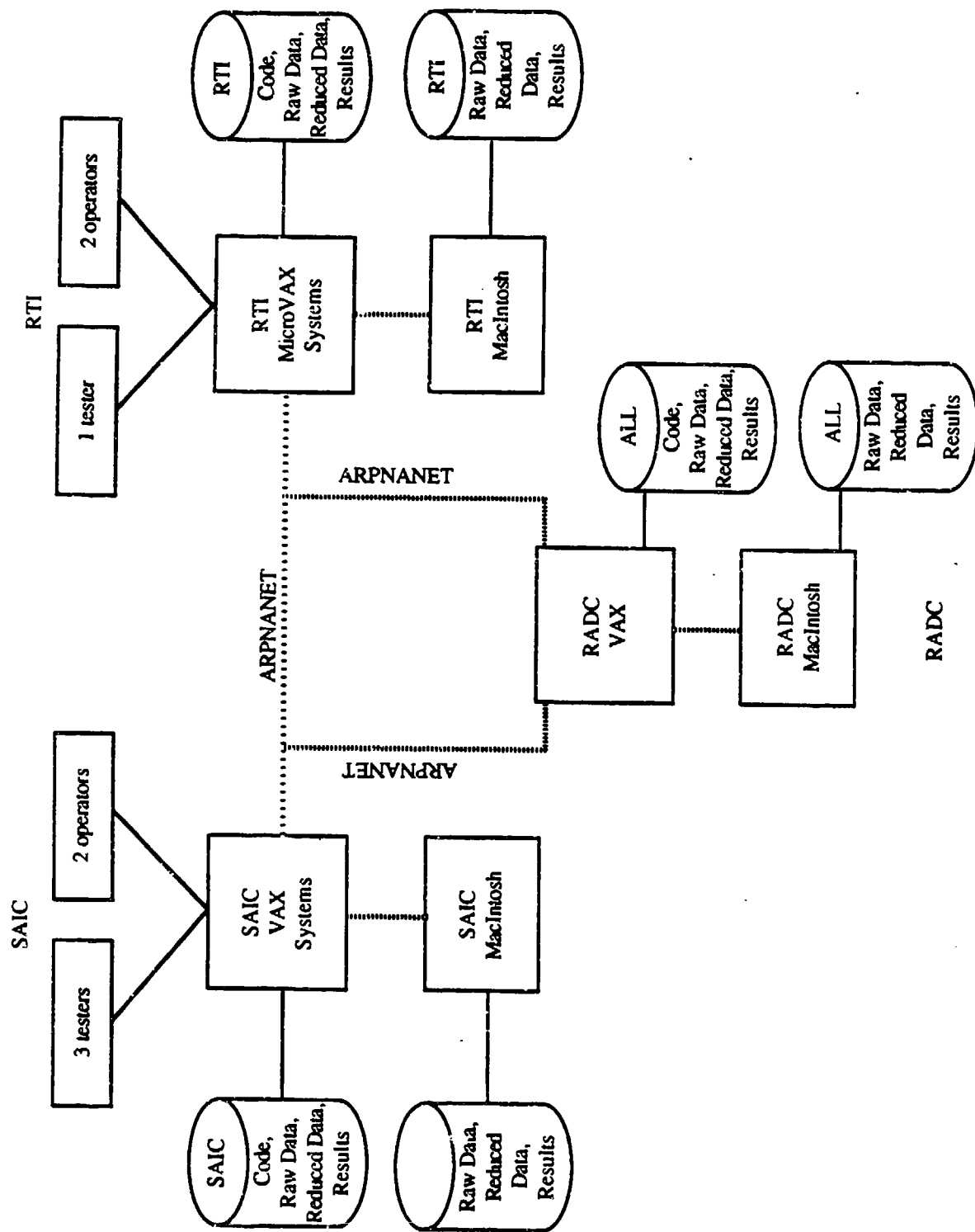


Figure 4.2 Software Test Configuration for Informal Testing

Table 4.2. Data Managers & Statistical Analyzers

TOOL	APPLICATION	SOURCE
4th Dimension	Manage test results	Acius
Statview	Data analysis on Mac II	Brainpower
SAS/GLM	Data analysis on VAX	SAS

Table 4.3. Software Test Support Tools

TOOL	APPLICATION	SOURCE
DTM	Manage software samples, test cases, test data.	DEC
SDDL	Support code reading.	SAIC
RXVP-80	Support error and anomaly detection, structural analysis, and branch testing. Also used to provide path coverage information.	GRC
Random Number Generator	Support Random Testing	RTI
Automated Comparator	Support Dynamic Test Analysis	SAIC
AMS	Code Analysis, Metrics	RADC
IRMS	Data Analysis, Results Data Base	S C

- b. Source code, enhanced with indentation based on structure and with flow line arrows in a two-dimensional representation.
- c. Module cross-reference listing.
- d. Module reference tree.

4.2.3.3 RXVP-80

RXVP-80 [22] was used to automate Structure Analysis and Error Anomaly Detection, to process code samples to identify branch coverage for the branch testing technique, and to instrument samples for path coverage information for all three dynamic techniques. The RXVP-80 test tool combines static and dynamic analysis features. Testers use its static features to obtain reports for the static analyses of the code samples as required for error and anomaly detection, structure analysis, and branch testing. They also instruct RXVP-80 to instrument the code under test with path coverage commands for use in dynamic testing. These operations are performed by running RXVP-80 independently of the DTM, whereas all dynamic testing takes place under the organization of the DTM. Thus code samples which are instrumented with the dynamic component of RXVP-80 are invoked from within a DTM test description template file.

The following steps summarize the use of RXVP-80 for all experiment testing.

- a. All Dynamic Techniques:
 - 1. Instrument each sample with RXVP-80 to obtain path coverage measurements.
 - 2. Gather and record path coverage for each test case and for all test cases in one technique application.
- b. Branch Testing
 - 1. Use RXVP-80 to produce source code listing with branches identified.
 - 2. Convert branches to paths and use paths to create test cases that will meet the stopping rule.

4.2.3.4 CPU Use Procedures

Two routines (begin & end CPU usage collection) written in FORTRAN were provided by RTI. These routines are called from the sample driver in order to measure CPU time used for each test case execution.

4.2.4 Software Code Samples

Test project software for the present experiment consists of available life-cycle versions of two selected test projects. Their documentation includes some or all of the following: system and software specifications; test plans, specifications and reports; standards and practices for specification, design, development, and test; and system and Software Trouble Reports. The project software and documentation are identified in Volume 3 of the *Task II Report* [8].

It was originally assumed that code samples could be selected for each testing level from archived tapes of actual code immediately prior to the test level of interest. This assumption was based on preliminary reports of project characteristics. However, when project tapes and documentation were received and evaluated, it became clear that the projects were tested on a functional build basis, rather than separate unit testing, CSC integration and testing, and CSCI testing. Thus, a given software build might include some untested units, some partially tested units, some partially complete or partially tested CSCs, and so forth. The sample selection strategy was then reworked to accommodate build testing.

The sample selection algorithm documented in the SRTIP for a given test level is as follows:

- a. Review error distributions among the four candidate projects and construct, to the extent possible, error-prone sample pools with (in decreasing priority):
 1. Similar average number of errors per component in each pool, and
 2. Project pool size of at least twice as many components as will be chosen as samples.
- b. Randomly choose which projects will be sampled, if more than enough qualifying sample pools are created.
- c. Randomly choose the necessary number of samples from each pool.
- d. Validate each sample by obtaining the earliest available version of the sample code and ensuring it contains the required number of errors (see step a.1)
- e. Repeat the steps a to d on a sample-by-sample basis for any sample(s) that fail step d.

This process yielded a set of unit samples from the Sim/Stim project which were sufficiently high in the calling structure that they could not easily be separated from the build environment to accommodate stand alone testing with an independent driver. This was unacceptable for our limited effort budget. A basis other than random selection from among the most error-prone code was needed which would yield independently testable units and CSC-level integrations of units that meet the following objectives:

- a. Samples must be selected in a uniform manner and have similar characteristics.
- b. Samples must consist of error-prone portions of code.

The known error profiles for these projects indicated that the error counts for selectable samples would be considerably lower than before, but hopefully they would be acceptable for our experiment. Accordingly, the following procedure was established to reselect code samples at the unit test and CSC integration and test levels.

- a. Utilize project consultants to identify a pool of units (and integrations of units) for each project:

1. Which can be tested independently from the software build.
2. Whose driver development will not exceed a few man-days of effort.
3. That contain as many known errors as possible.

- b. Select the four samples from each pool which have the most errors.

Table 4.4 and 4.5 identify, respectively, the code samples and Latin Squares from the Sim/Stim and SCENE projects which were utilized during unit and CSC integration testing levels.

4.2.5 Personnel

SAIC's experience in applied software reliability measurement and testing techniques is complemented by RTIs research work in the field of software test experiment design and analysis. Both of these strengths have been augmented by contributions during Task 2 of carefully selected consultants with knowledge of the test techniques and the experiment design. Additional consultants were utilized during the experiment conduct; each is an identified expert on one of the selected software projects. Specific personnel qualifications for each of the experiments and empirical studies were as follows:

- a. Activity Leader (1). Key project person; responsible for activity assignment, coordination, completion, and products.
- b. Assistant Activity Leader (1). Key project person -- assists Activity Leader.
- c. Tester (4). Programmer/test and evaluation specialist -- produces test descriptions and test procedures; develops test drivers; performs tests for all test techniques; analyzes test outputs and documents observed test errors; records other required test data.
- d. Consultant (2). Test project specialist; assists in preparations for all testing; documents supplemental requirements as needed for selected code samples and test drivers; verifies test results.
- e. Data Entry (2). Administrative staff; enters test results data into the database.
- f. Analyst (3). Data analysis specialist; ensures test data is pure; reduces test data for analysis; produces descriptive analyses of raw and tabulated data.

4.2.6 Tester Instructions

A special set of instructions [11] were prepared to guide testing. These instructions provided information necessary for testers to conduct their tasks during experimental testing activities. The testing activities were part of an experimental framework, so were geared toward meeting two goals of the experimental study:

- 1) Ensuring each tester follows set procedures as closely as possible, so that he/she conducts each testing technique in the same controlled manner;
- 2) Ensuring all data and other deliverables obtained are consistent and complete.

Table 4.4. Unit Sample Characteristics

Project	Sample Characteristics		
	Sample ID	No. of SPRs (original)	Tape Source
Sim/Stim	SNWMSN	8	Build 1
	THXRDT	2	Build 1
	THUADC	3	Build 1
	THUSCN	3	Build 1
SCENE	SCANNER	2	Vers. 6.0
	INFANSEN	4	Vers. 6.0
	INPSAT	4	Vers. 6.0
	INMODE	3	Vers. 6.0

Table 4.5. CSC Sample Characteristics

Project	Sample Characteristics		
	Sample ID	No. of SPRs (original)	Tape Source
Sim/Stim	SNWMSN	18	Build 1
	THURDT	4	Build 1
	THUADS	6	Build 1
	THPCON	5	Build 1

These instructions were meant to encompass all information that testers and operators would need during the software testing. Respective roles of testers and operators are defined. (In actual application, operators were minimally utilized and testers performed most of prescribed roles. Activities within those roles include all aspects of testing software samples using six different test techniques (branch testing, code review, error and anomaly detection, functional testing, random testing, and structure analysis) at two test levels (unit and CSC). A description of each test activity, the order of performing each activity, the resources needed (inputs) to accomplish the tasks, and the results (outputs) required at completion of those tasks were provided.

In addition, instructions were provided outlining how to obtain answers to questions and solutions to problems encountered during these activities. A schedule for the entire testing task, and references for various aspects of the activities, e.g. test techniques, test support tools, were provided.

Chapter 1 of the tester instructions provides an overview of the purpose and scope of the manual. Chapter 2 identifies the six test techniques to be applied by the testers, and provides a high-level definition of each technique. Chapter 3 identifies and describes usage of the resources available to this effort, including computing systems, test support tools, forms, and management.

Chapter 4 documents procedures which testers and operators must follow in performing their work for the experiment. These procedures range from the test environment configuration stage to the documenting results and activities phase. This section is in essence a tutorial; step-by-step procedures are given for each test technique. All rules, such as required order of test level execution, required order of test technique application within a level, and assignment of samples to testers are provided.

Chapter 5 includes a test schedule for each test level. Attachment A of the instructions is unique for each tester. It identifies which samples the tester will test at each test level. It also provides a required ordering in which the test techniques must be applied to each assigned sample. Attachment B contains all of the test data collection worksheets to be utilized to collect the necessary test specifications and results. Detailed instructions are provided there to complete each form. Attachment C was developed as the pilot experiment was performed. This attachment includes some 23 Tester Information Bulletins which were issued to clarify and/or supplement information contained in the original instructions.

4.3 Testing Techniques

Eighteen test techniques are represented in the STH, including some that are not commonly used in practice but are currently used in research (for example, mutation testing). Still newer state-of-the-art test techniques were identified by testing experts acting as consultants.

While including such promising techniques in this experiment would have provided some much-desired data for the testing community, it was beyond the scope of the study, given that there were already more methods addressed in the STH than we could include in our effort. Further experiments of this type utilizing additional techniques are recommended.

Table 4.6. Testing Techniques and Automated Tools

TESTING TECHNIQUE	FUNCTIONAL SUMMARY	TEST TOOL UTILITIES
Branch Testing	Force execution of all possible program branches; detect deviations from expected program outputs.	RXVP-80
Functional Testing	Determine if the program performs the intended functions fully and correctly.	(None)
Random Testing	Utilize random combinations of inputs to isolate incorrect algorithms and computations.	Random number generator
Code Review	Visually inspect a program using a checklist in order to identify types of faults.	SDDL
Error and Anomaly Detection	Perform syntax, data and physical units checking to detect program faults.	RXVP-80
Structure Analysis	Evaluate control and code structures and subprogram usage to detect program faults.	RXVP-80

Early in the present study, a survey was conducted to provide an analysis of the use, advantages, and limitations of state-of-the-art testing techniques. This survey was conducted by reviewing various reference documents and reviewing test documents from different projects. A list of these reference documents and test documents is presented in the *Task 1 Report* [5].

The following testing techniques were selected from the referenced study, and are evaluated in the present report:

- a. Static Test Techniques
 - 1. Code Reviews
 - 2. Error and Anomaly Detection
 - 3. Structure Analysis
- b. Dynamic Test Techniques
 - 1. Branch Testing
 - 2. Random Testing
 - 3. Functional Testing

Table 4.6 presents an overview of these testing techniques, and the software test tools that were used to support and/or automate them. Four of the techniques (code review, functional, random and branch testing) were performed as part of the experimental study by all four testers at each test level. The remaining two techniques (error & anomaly detection and structure analysis) were performed as an empirical study by only two of the testers at each test level. See section 4.5 for detail on the experiment design.

This section defines each testing technique and gives a high-level description of inputs to testing with a given technique, outputs from testing with the technique, and an overview of the process of applying that technique. For each test technique, the definition and description given has been customized for application in this study. For example, the specific commercial test support tools used are listed as inputs. Also, completed test data collection forms are listed as outputs; these forms are not innate to the test techniques, but are completed at precise points in the test processes of this study to standardize data collection for post-study analyses. Detailed instructions for applying each technique at the appropriate test levels are provided in the tester instructions.

This information has been compiled with the analysis of the test results (Chapter 5) and associated testing strategy recommendations (see Chapter 6) into a useful Guidebook (Volume 2) for acquisition managers and software engineers.

4.3.1 Dynamic Techniques

Dynamic test techniques are those which involve executing the code in the testing process. Three of the test techniques in this study are dynamic techniques: branch, functional, and random test techniques.

Black box techniques are those which do not involve knowledge of the source code; instead, more abstract knowledge of the problem, such as that provided in functional specifications and/or design documents, is used to create tests. *White box techniques* are those which require knowledge of the source code; tests for the software are designed based on knowledge of the characteristics of the specific implementation.

Two of the dynamic techniques, random and functional, are black box testing techniques, whereas branch testing is a white box testing technique.

Each technique is applied at the unit and CSC test levels. All three techniques are summarized below in terms of their inputs, process, and outputs.

Note that all dynamic techniques require an abstract specification of the program function as an input. According to DoD-STD-2167A, the specifications applicable at the unit and CSC test levels are:

- a. CSC level: Software Top Level Design Document (STLDD) or equivalent.
- b. Unit level: the Detailed Design Document (DDD) or equivalent.

For this study, equivalent specifications were produced by consultants knowledgeable of the projects. For CSC integration samples, the consultants identified a subset of the CSC functions to be tested within the budget and time constraints for the study. The consultants also provided requirements for the necessary test drivers.

4.3.1.1 Branch Testing

This testing technique combines static and dynamic techniques and detects failures. The static portion is used to determine test data that force selected branches to be executed. The dynamic portion is used to actually run the code with these test data and then obtain test outputs. Test personnel perform branch testing of the code samples using the appropriate documentation and specifications. Test coverage analysis is used to detect untested parts of the program. Output data analysis is applied to detect outputs that deviate from known or specified output values.

Procedure

Branch testing requires creating test cases that cause execution of a specified percentage of all branches in the code. As applied in this effort, Branch testing consists of two subset of techniques defined in the RADC Software Test Handbook. These two techniques are 1) static input space partitioning by path analysis, and 2) dynamic instrumentation-based path and structural analysis.

The first technique subset identifies all branches in the code under test. The tester uses this information to create tests that will cause execution of each branch. RXVP-80 is used to identify all of the branches in the code. The tester determines the correct outputs expected from executing the code for each test case based on the specifications. Then, the tester writes a test procedure for these test cases, tailors the test driver as needed, and sets up the DEC Test Manager (DTM) test collections and descriptions.

The second technique subset is used to execute the code under test with the tests created, and to track the branch coverage for each test case and for the total of all test cases. RXVP-80 is used to instrument the code under test. The tester executes the code with the given test case inputs by running the DTM test collection, and uses the DTM Review capability to find differences in actual and expected outputs. Execution of the sample code instrumented by RXVP-80 generates reports on branch coverage. Testers review these reports to ensure that their test cases meet the stopping rule; if they don't, the tester returns to the static activity of creating more test cases to

execute all branches at least twice. Testers compare expected outputs with outputs obtained by dynamic testing, and log all errors found on SPRs.

Information Input

Inputs to the static portion of branch testing are: 1) the source code of the sample and 2) the specifications of interest for the sample, and (3) the RXVP-80 tool.

Inputs to the dynamic portion are: 1) the source code of the sample, 2) the test cases and test procedures generated during the static portion of branch testing, and 3) the RXVP-80 tool.

Information Output

Outputs from the static portion of branch testing are: 1) test cases, 2) test procedures, and 3) test activity worksheets.

Outputs from the dynamic portion are: 1) the actual outputs, from executing the sample with each test case, 2) SPRs which document errors discovered in those outputs, 3) test case branch coverage reports, and 4) test activity worksheets.

4.3.1.2 Functional Testing

This dynamic testing technique finds failures consisting of discrepancies between the program and its specification. In using this testing technique, the program design is viewed as an abstract description of the design and requirement specifications. Test data are generated, based on knowledge of the programs under test and on the nature of the program's inputs. The test data are designed to ensure adequate testing of the requirements stated in the specification. Test personnel perform functional testing of the code sample using the appropriate documentation and specifications. Functional testing is performed to assure that each unit correctly performs the functions that it is intended to perform.

Procedure

Functional testing entails creating test cases to exercise all functions, or a given percentage of all functions, that the software specifications include as functional requirements. The tester consults the appropriate functional specifications provided, and manually creates test cases and corresponding test procedures to test all applicable functions. The test driver is tailored as needed, and the DTM is set up to run the tests as a test collection. The test sample is instrumented with RXVP-80 in order to gather path coverage information; note that this is not integral to functional testing, but is done to provide path coverage data of functional tests for experimental analyses. The tester executes the code with the given test case inputs by running the DTM test collection, and uses the DTM Review capability to find differences in actual and expected outputs. SPRs are logged for all errors found.

Information Input

Inputs to functional testing are: 1) an abstract specification of the program function and 2) the source code.

Information Output

Outputs of functional testing are: 1) test cases, 2) actual test outputs, 3) an SPR for each error found, and 4) test activity worksheets.

4.3.1.3 Random Testing

This dynamic testing technique tests a unit by randomly selecting subsets of all possible input values. The input subsets can be sampled according to the actual probability distribution of the input sequences, thus characterizing the usage; or according to other probability distributions. Random testing invokes unusual combinations of input values that are frequently not represented in test data generated by using the other test techniques. Random testing is performed to detect outputs from the random inputs that deviate from known or expected output values. Test personnel perform random testing of code samples using the appropriate documentation and specifications. Papers by Duran and Ntafos [19] and Bell [20] provide more information on random testing.

Procedure

Random (or statistical) testings consists of randomly choosing test cases as subsets of all possible input values according to a uniform probability distribution over the range of values specified for each input. Working from code sample specifications which identify all valid inputs and outputs, the testers code a test generator routine that randomly selects inputs. This generator is then executed to provide test case inputs. Testers determine the corresponding correct outputs expected. Then test cases are prepared from these test input and output pairs, and a test procedure is written to execute them.

The test driver is tailored as needed and the DTM is set up to run the tests as a test collection. The sample is instrumented with RXVP-80 in order to gather path coverage information. The tester executes the code with the given test case inputs by running the DTM test collection, and uses the DTM Review capability to find differences in actual and expected outputs. SPRs are logged for all errors found. If the stopping rule has not been met, the tester returns to the static activity of creating more test cases to achieve the required MTTF of 10 input cases.

Information Input

Inputs are: 1) the source code of the sample; and 2) an abstract specification, or equivalent, of the sample functions, including range specifications of inputs and outputs.

Information Output

Outputs are: 1) a set of test cases and corresponding test procedure(s), 2) a completed SPR for each error observed, and 3) test activity worksheets.

4.3.2 Static Techniques

Static test techniques are those which do not involve executing the code in the testing process; instead, the source code is inspected or reviewed. The remaining three test techniques in this study are static techniques: code review, error and

anomaly detection, and structure analysis. All three are white box techniques, as defined earlier in Section 4.3.1. These techniques are applied at the unit and CSC test levels and are described below.

4.3.2.1 Code Review

This static testing technique involves the visual inspection of a program with the objective of finding faults. Test personnel perform an inspection of the sample code units using the appropriate documentation and specifications. These code inspections are driven by checklists in order to identify common types of errors in the code.

Process

Initially, the source code is statically processed by the SDDL tool. Thus the source is enhanced with indentation based on structure logic and with flow line arrows in a two-dimensional representation. Other outputs of SDDL for organizational use in the code inspection are the table of contents, a module cross reference listing, and a module reference tree. The tester works through the checklist sequentially, referring to the annotated source listing and above mentioned organizational aids.

The tester looks for errors (and often for poor programming practices) in the source code and comments by examining it for attributes noted in the checklist. This checklist identifies all aspects of the code to be studied for problems and all checks to be made for agreement between the code and the specifications. Examples of code attributes in the checklist are: 1) whether branch points and loop indices have been coded correctly; 2) whether formal and actual parameters are consistent and correct; and 3) whether specifications, inline comments, and code are consistent and code is complete with respect to the specifications. When a problem or error is found during the code review, an SPR is completed.

Information Input

Inputs to code review are 1) the sample source code, 2) the code review checklist, and 3) the relevant specifications.

Information Output

Outputs from code review are: 1) an SPR for each error found, and 2) test activity worksheets.

4.3.2.2 Error & Anomaly Detection

This static testing technique is applied to detect faults via syntax checking, data checking and physical units checking of the source code. Syntax checking includes checks for uninitialized variables and unreachable code. Data checking entails identifying set-use anomalies, conducting a data flow analysis and unit consistency checking. Physical unit checking looks for consistency in physical units usage. Test personnel perform error and anomaly detection on the selected sample code units using the appropriate documentation and specifications.

Process

Error and anomaly detection is applied using the following static analysis functions of the automated tool RXVP-80:

- a. **Syntax checking:** uninitialized variable screening, and unreachable code screening.
- b. **Data checking:** data flow set/use anomalies, interface completeness and consistency (CSC integration level).
- c. **Physical Units Checking:** checking for consistency in physical units (e.g. feet, gallons, liters, etc.) usage.

An SPR is completed for each error found; some errors may be reported directly in the RXVP-80 reports, some may be found by visual inspection of the code, and some by other means during the error and anomaly testing activities.

Information Input

Inputs to error and anomaly detection are: 1) the source code to be analyzed, 2) the specifications, and 3) the RXVP-80 testing tool.

Information Output

Outputs from error and anomaly detection are: 1) an SPR for every error found, and 2) test activity worksheets.

4.3.2.3 Structure Analysis

This static testing technique detects faults concerning the control structures and code structures of FORTRAN, and improper subprogram usage. Test personnel perform structure analysis on the source code using the appropriate documents and specifications. Structure analysis is performed to determine the presence of improper or incomplete control structures, structurally dead code, possible recursion in routine calls, routines which are never called, and attempts to call non-existent routines.

Process

The automated tool RXVP-80 is used to partially automate structure analysis. RXVP-80 processes the source code and parameters describing control flow standards, and provides error reports and a program call graph. For example, analysis of graph cycles may indicate unintentionally recursive code; presence of a disjoint subset of the graph illustrates unreachable, or dead, code; and calls to nonexistent routines will be illustrated by edges with no sink nodes. The tester logs RXVP-80 error reports and errors illustrated by the call graph in SPRs. Any errors found by other means also are logged in SPRs.

Information Input

Inputs to structure analysis are: 1) the source code; 2) a specification of the control flow standards to be enforced in the language; and 3) the RXVP-80 tool.

Information Output

Outputs from structure analysis are: 1) an SPR for every error found, 2) a program call graph or report, and 3) test activity worksheets.

4.4 Reliability Estimation

Measurement data for software reliability estimation was collected as an integral part of the software testing experiment since reliability estimation is based on performance results during test conditions. Once software is executing a failure rate can be directly observed, thus avoiding a transformation of fault density necessary in determining failure rate during pre-test software reliability prediction. The failure rate of a program during test is expected to be affected by the amount of testing performed, the methodology employed, and the thoroughness of the testing. An evaluation of each of these factors is included in the experiment design.

4.4.1 Reliability Estimation Number (Model 2)

The Reliability Estimation Number (REN) is an Estimated Failure Rate (F). REN Model 2 is specified in Task 201 of the SRPEG and provides the basis for our experimental evaluation of software reliability estimation. It uses the failure rate observed during testing and modifies that rate by parameters estimating the thoroughness of testing and the extent to which the test environment simulates the operational environment. Tables 4.7 and 4.8 identify these REN data elements and procedures and their respective data collection sources and metric worksheets, as specified for Task 201.

Utilizing the SRPEG procedures, two RENs are computed for the code samples tested during the unit and CSC test levels for the dynamic test techniques only. They are referred to as REN_AVG (based on average failure rate during test) and REN_EOT (based on failure rate at end of test). Computation of these RENs is presently infeasible for the static test techniques. We were unable to compute REN_EOT due to the low levels of testing and the design decision to not repair detected errors.

REN_AVG and REN_EOT, also referred to as estimated failure rates (F), are computed as follows:

$$\begin{aligned}\text{REN_AVG} &= \text{FT}_1 * \text{T}_1 \text{ or} \\ \text{REN_EOT} &= \text{FT}_2 * \text{T}_2\end{aligned}$$

where: FT_1 is the average observed failure rate during testing.
 FT_2 is the observed failure rate at end of test.

$$\begin{aligned}\text{T}_1 &= .02 * \text{T} \\ \text{T}_2 &= .14 * \text{T} \text{ and} \\ \text{T} &= \text{TE} * \text{TM} * \text{TC}\end{aligned}$$

where: TE is a measure of Test Effort
TM is a measure of Test Methodology
TC is a measure of Test Coverage

Table 4.7. REN Data Collection Procedures

METRIC DATA (TASK 201)	SRPEG DATA COLLECTION PROCEDURE
Average Failure Rate During Test (F_{T1})	12, 13, 14
Failure Rate at End of Test (F_{T2})	12, 13, 14
Test Effort (TE)	15
Test Method (TM)	16
Test Coverage (TC)	17

Table 4.8. REN Data Sources

METRIC DATA	INPUT DOCUMENTS	SRPEG METRIC WORKSHEETS
F_{T1}	SPRs OS Reports Tester Logs	5, 6
F_{T2}	SPRs OS Reports Tester Logs	5, 6
TE	Tester Logs	6
TM	Test Plans Test Procedures Software Development Plan Software Test Handbook	7
TC	Source Code Test Plans Test Procedures Requirements Document	8

4.4.2 REN for Test Environments

The influence the test environment has on the estimated failure Rate (F) is described by three metrics. These metrics are in the form of multipliers. The product of all of these metrics is used to adjust the Observed Failure Rate (FT) up or down depending on the level of confidence in the representativeness and thoroughness of the test environment.

4.4.2.1 Average Failure Rate During Testing (FT1)

FT1 can be calculated at any time during testing. It is based on the current total number of SPRs recorded and the current total amount of test operation time expended. It is expected that the failure rate will vary widely depending on when it is computed. For more consistent results, average failure rates are calculated for each test phase.

4.4.2.2 Failure Rate at End of Testing (FT2)

FT2 is based on the number of SPRs recorded and amount of computer operation time expended during the last three test periods of testing.

4.4.2.3 Test Effort (TE)

Three alternatives are provided for measurements TE and are based upon data availability:

- a. The preferred alternative is based on the labor hours expended on software test. As a baseline, 40 percent of the total software development effort should be allocated to software test. A higher percentage indicates correspondingly more intensive testing, a lower percentage less intensive testing.
- b. The second alternative utilizes funding instead of labor hours.
- c. The third alternative is the total calendar time devoted to test.

Calculate TE, based on these three characteristics, as follows:

TE = .9 if $40/AT \leq 1$, or

TE = 1.0 if $40/AT > 1$

where: AT = the percent of the development effort devoted to testing.

4.4.2.4 Test Methodology (TM)

TM represents the use of test tools and test techniques by a staff of specialists. The STH specifies a technique to determine what tools and techniques should be applied to a specific application. That technique results in a recommended set of testing techniques and tools. The approach is to use that recommendation to evaluate the techniques and tools applied on a particular development.

Calculate TM as follows:

TM = .9 for $TU/TT \geq .75$
TM = 1 for $.75 > TU/TT \geq .5$
TM = 1.1 for $TU/TT < .5$

where: TU is the number of tools and techniques used.
TT is the number of tools and techniques recommended.

4.4.2.5 Test Coverage (TC)

TC assesses how thoroughly the software has been exercised during testing. If all of the code has been exercised then there is some level of confidence established that the code will operate reliably during operation. Typically however, these programs do not maintain this type of information and a significant portion (up to 40%) of the software (especially error handling code) may never be tested.

Calculate TC as follows:

$$TC = 1/VS$$

where: VS = VS1 during unit or CSC testing
VS = VS2 during CSC integration and test, and

$$VS1 = (PT/TP + IT/IT)/2$$

where: PT = execution paths tested
TP = total execution paths
IT = inputs tested
TI = total number of inputs

$$VS2 = (MT/NM + CT/TC)/2$$

where: MT = units tested
NM = total number of units
CT = interfaces tested
TC = total number of interfaces

4.5 Experiment Design and Conduct

The design adopted for this study is presented here and in the SRTIP. It is specifically geared toward providing results useful to acquisition managers of Air Force software. To ensure a sound, comprehensive design, the initial design was enhanced with inputs and reviews by RTI experts in experiment design with human subjects and statistics and enhanced with inputs and review by industry and academic experts in the fields of computer science, software testing, experiment design with human subject, statistics, and software metrics [17].

As illustrated in Figure 4.3, an adaptation of a figure developed by Basili and Reiter [18], a standardized approach to the experiment design, was taken. In following this approach, a precise definition of experimental goals in the form of specific questions to be answered was developed from the objectives declared in the SOW and the contents of the input guidebooks. These questions were organized into "questions of

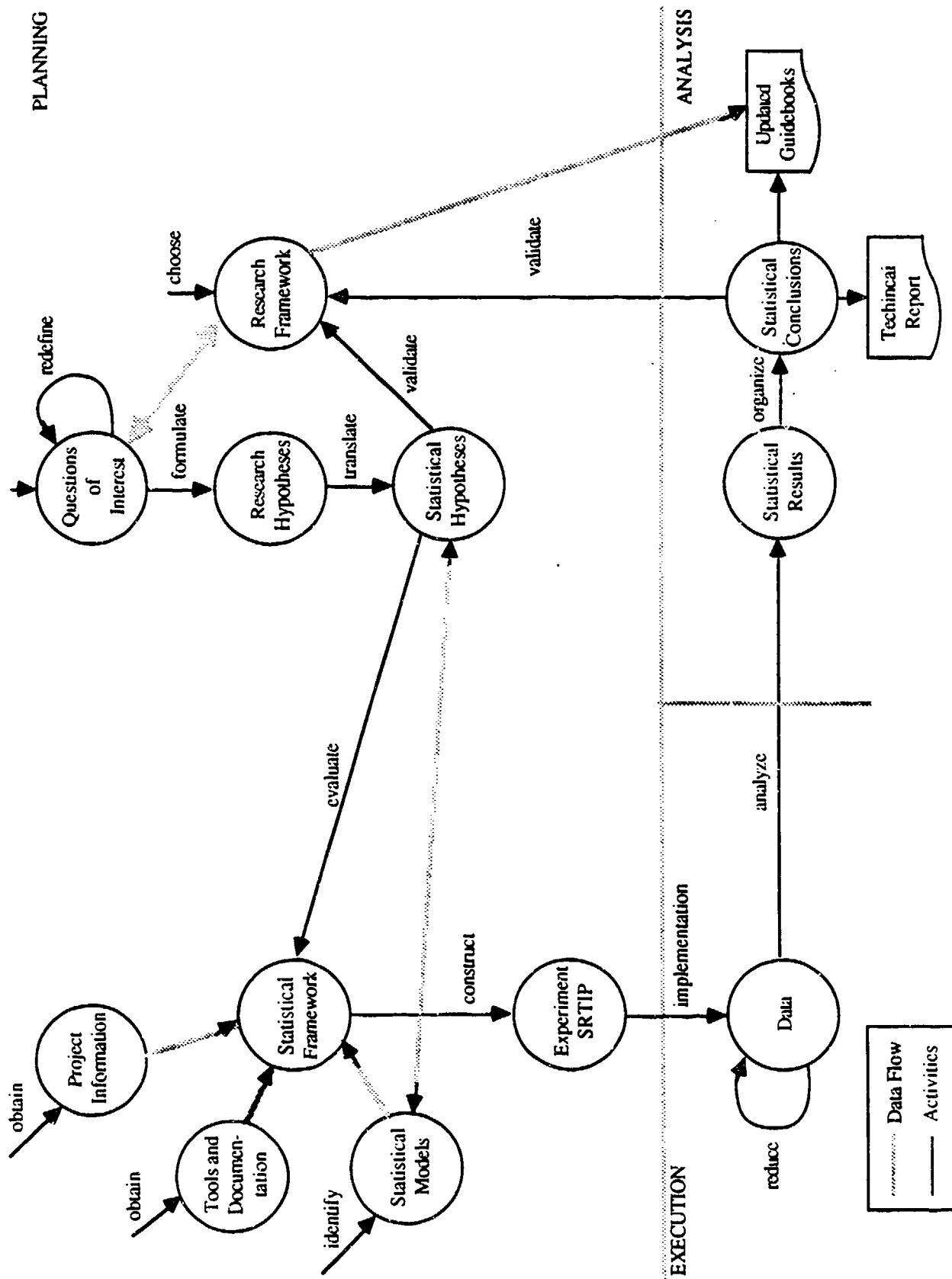


Figure 4.3 Approach to the Experiment Design

interest" and are documented in the SRTIP. Candidate statistical designs were evaluated in terms of how well they would test the chosen questions of interest, given the projects, test techniques, tools and other resources available.

Experiments to address all aspects covered in the two input guidebooks were beyond the scope and resources of this study. Thus, the design was tailored to address as much of the information in these two documents as possible, while controlling the experimental variables to the extent necessary to preserve the statistical soundness of the design.

A combination of experiments and empirical studies were selected to best meet the objectives of this study. The empirical studies address deterministic procedures, so only two testers are required to apply each test technique chosen for the empirical study. The experiments address nondeterministic procedures and employ four testers. Samples consist of code from actual software product development.

During experimentation, the software project code samples were tested using six different test techniques. Measures were obtained for unit and CSC integration levels of testing. The test techniques studied were random testing, functional testing, branch testing, code review, error and anomaly detection, and structure analysis. The study obtained quantitative measures of software test effectiveness and efficiency during the testing process.

The experiment was performed initially as a small-scale pilot experiment. The overall experiment design and methodology was verified in the pilot. This step ensured the validity of the remaining experimentation and attendant findings and recommendations. Data was collected to provide insight into the effectiveness of test techniques for different application types at different test levels, per the test objectives. Included are the test effort required, test coverage goals, and software error characteristics.

4.5.1 Experiments

Experiments were designed for each of the four deterministic testing techniques. These experiments address:

- a. A study of three dynamic test techniques: functional, random, and branch testing. Data analysis is by percent known errors observed at each test level, by percent known errors observed by test technique, and by percent known errors observed by test technique at each test level. (Absolute number of errors are recorded for descriptive purposes. Since each sample does not have the same total number of errors, meaningful comparisons across samples are made on a percentage basis.)
- b. A study of one static test technique: code review. Data analysis is by percent of known errors detected, by percent of known errors detected at a test level.
- c. The measurement of each technique's relative efficiency, in terms of the number of discrepancy reports filed, on a cost and time reference basis.

The experiments are grouped by test level; a portion of the unit test level experiment is run first, as a pilot study, followed by the remainder of the unit testing, then CSC integration and test experiments.

Since the experimental test techniques are nondeterministic, different testers design different test cases when applying a technique to a given sample. Four experienced testers are utilized to minimize differences among testers' abilities and the amount that chance plays in the creation of a given test case. Each tester uses all four techniques on his or her assigned code samples. This is done at each test level, and techniques are applied in a predefined sequence utilizing Latin Squares, thus assuring that every technique precedes every other technique an equal number of times.

A feature of the Latin Square design is that every row and every column is a complete replication; that is, in every row and in every column, each test method is applied exactly once. This grouping has the benefit of removing from the generic error term all differences among rows (test periods) and all differences among columns (software sample/tester pairs).

The Latin Squares used in this design were specifically chosen so that the treatments (test techniques) are balanced with respect to residual effect in the sense that every treatment is preceded by every other treatment an equal number of times (once per square). Thus, the residual effects, or the learning about the software and its faults or errors that the tester may experience by applying a test technique to that software, are evenly distributed. If a tester creates better test descriptions for functional testing, and if he has already created the test descriptions for branch testing, that effect is not hidden among all experimental runs, since the order of testing is varied in a controlled manner. This setup also allows measurement of the combined tester/sample effect.

4.5.1.1 Unit Test Level

Four testers were employed at the unit test level. Both software projects have units preserved on build tapes. The Sim/Stim and SCENE were chosen for use at this level because their sample pools were the most error-prone units for each project, as described in Section 4.2.4. Table 4.9 identifies the test samples utilized from Sim/Stim and SCENE. One sample from each project was randomly assigned to each tester. Each tester applied each test technique to each of his or her samples.

A predetermined order in which the tester applied each test technique was assigned by Latin Square experiment design (as denoted by the Session designation in Table 4.9). This defines the order in which the tester created test descriptions, and then test procedures, for that sample. It is also the order in which testers evaluated test outputs for errors.

As shown in Table 4.9, the unit level experiment has been displayed in two Latin Squares. In each Latin Square, the row is associated with the order of testing and the columns are associated with testers and software types. The square using the SCENE project comprised the remainder of the unit project.

4.5.1.2 CSC Integration Test Level

Four testers were employed at the CSC integration and test level. Sim/Stim was used at this level. Four CSC unit integration samples were chosen from a pool of error-prone CSCs, as described in Section 4.2.4. Table 4.10 identifies the samples utilized. One sample was randomly assigned to each tester. Each tester applied each test technique

to his or her sample, in the order prescribed by the design. Table 4.10 provides this ordering.

4.5.2 Empirical Studies

Empirical studies were designed for both of the nondeterministic testing techniques. These empirical studies address:

- a. A study of two static test techniques: structure analysis and error and anomaly detection. Data analysis is by percent of known errors detected, by percent of known errors detected at a test level.
- b. The measurement of each technique's relative efficiency, in terms of the number of SPRs filed (errors located), on a cost and time reference basis.

As shown in Table 4.11, empirical studies were conducted at the unit and CSC test phases using the two deterministic, static test techniques: structure analysis and error and anomaly detection. Two testers applied each technique to the same code sample which was utilized in the experiments in Section 4.5.1, as shown in Table 4.11. Note that since the test techniques are deterministic, only two testers were required and the order of test technique application is unimportant. The first two testers who completed their experiment testing duties were employed as the empirical study testers.

4.5.3 Pilot Experiment

A pilot study was performed prior to the complete set of experiments. This initial experimentation was conducted on the AFATDS Sim/Stim unit code samples in Tables 4.9 according to the established design and protocol. The pilot study is a small, integral portion of the entire experiment.

The experimental process and products were closely monitored during the pilot, and the knowledge gained was applied to alter and improve the process for the extended experimentation. Specifically, the results of the pilot were originally intended to:

- a. Fine tune the protocol. Tester Information Bulletins (TiBs) were instituted as a mechanism to clarify the tester instructions (see Section 4.5.4).
- b. Further reduce any remaining tester learning effects related to the test techniques, test tools, and hardware and software environments. No further refinements to the design were needed.
- c. Provide initial data on whether significantly different results are obtained at the two different test sites. Test results for the single tester at RTI correlated with similar results for the three testers at SAIC. This data is incorporated in Chapter 5.
- d. Provide initial data on how many of the known errors are found by the testers employing each of the prescribed test techniques. This was reported at the pilot briefing and is incorporated in Chapter 5.
- e. Provide initial data on whether significantly different results are obtained from the different testers working on different samples. It was

Table 4.9. Unit Test Latin Squares

Pilot Latin Square:

Sim/Stim PROJECT				
Unit	THUSCN	SNWMSN	THUADC	THXRDT
Tester	I	II	III	IV
Session 1	F	C	R	B
Session 2	C	R	B	F
Session 3	B	F	C	R
Session 4	R	B	F	C

Unit Latin Square:

SCENE PROJECT				
Unit	SCANNER	INFANSEN	INPSAT	INMODE
Tester	I	II	III	IV
Session 1	F	B	R	C
Session 2	B	R	C	F
Session 3	C	F	B	R
Session 4	R	C	F	B

B denotes the branch testing technique.

C denotes the code review testing technique.

F denotes the functional testing technique.

R denotes the random testing technique.

I, ..., IV are tester identifications. Each tester must be a unique individual.

Session denotes the order in which a tester applies the prescribed test method to the assigned sample.

Table 4.10. CSC Test Latin Square

Sim/Stim PROJECT				
CSC	THURDT	THUADS	SNWMSN	THPCON
Tester	I	II*	III	IV
Session 1	C	F	B	R
Session 2	R	B	C	F
Session 3	B	R	F	C
Session 4	F	C	R	B

B denotes the branch testing technique.

C denotes the code review testing technique.

F denotes the functional testing technique.

R denotes the random testing technique.

I,...,IV are tester identifications. Each tester must be a unique individual.

* actually tested by Tester III because of Tester II unavailability.

Session denotes the order in which a tester applies the prescribed test method to the assigned sample.

Table 4.11. Empirical Study Design

Unit Phase			CSC Phase	
Software	AFATDS	SCENE	Software	AFATDS
Tester X	S,E	S,E	Tester X	S,E
Tester Y	S,E	S,E	Tester Y	S,E

S denotes the static structure analysis technique.

E denotes the static error & anomaly detection test technique.

AFATDS is a unit code sample from project AFATDS Sim/Stim.

SCENE is a unit code sample from project SCENE.

X & Y denote first two testers to complete experimental testing.

found that there were considerable differences in test results between different testers/samples. This data is incorporated in Chapter 5.

Several additional evaluations were made of the pilot experiment process, each resulting in a practical refinement to the experimental testing procedure. No revisions to the formal experiment framework were needed. Changes that were made are:

a. Mid-Pilot changes

1. 23 TIBs were issued:
 - a) Worksheets and instructions were clarified.
 - b) A uniform Code Review Checklist was extracted from the original list of code checks and rational.
 - c) Stopping Rules were restored to revised estimates in the SRTIP.
 - d) Tool use and anomaly reporting was clarified.
2. Operator Roles were deleted per tester choice, as it was more efficient for testers to perform those activities themselves.

b. Post-Pilot Changes

1. The Code Review Checklist was split into two lists (unit and CSC) and reverted to less formal 'guidelines' to accommodate the testing technique objectives for applicable test levels.
2. DTM utilization was dropped following unit testing since its use was time consuming and there was no experiment design requirement to benchmark expected outputs.

4.5.4 Communications and Monitoring

During the conduct of the experiments and empirical studies at SAIC and RTI, communication lines were maintained to ensure that all questions regarding setup, procedure, and so forth were resolved quickly, and to keep the experimental environment and conditions as similar as possible across the two sites. This communications scheme supported rapid resolution of questions and provided electronic, telephone, and mail paths to both sites and all test personnel.

Any experiment-related issue, or question that was initiated by testers, was investigated by the activity managers at both sites. When an answer was obtained, it was documented in a Tester Information Bulletin (TIB), then distributed and filed at both sites. This included the resolution of questions which had any impact on instructions to testers.

The experiment documentation produced by testers was subjected a quality assurance (QA) review by the pilot and extended experiment activity managers for acceptability. Activity managers provided testers with scheduled review points; when these points were reached, testers informed the activity manager that materials were ready for review. The activity manager then checked the materials

for adherence to the standards set forth in the guiding documents and for completeness. These guidelines for determining acceptability were derived from the SRPEG and 2167 DIDs for test descriptions and test procedures. They were distributed to testers as instructions for completing the test data collection worksheets. All materials were reviewed in a timely fashion for appropriate content.

In addition to scheduled QA monitoring, regular evaluation of the general testing activities was conducted by the activity managers. Any irregularities were corrected as they occurred by applicable TIBs.

4.5.5 Applying the Techniques

Figure 4.4 illustrates the various facets of testing technique application. Essentially, testers applied each testing technique in the order specified by the Latin Squares, utilizing the procedures contained in the tester instructions. Test project consultants assisted in pre-test preparation and post-test evaluation activities. Testers created test cases, test procedures and test drivers from specifications provided by the consultants. A test harness was setup utilizing DTM to provide inputs to the test driver and to capture and compare test outputs from the code samples under test. During unit testing DTM was found to be of no actual assistance for this type of experimentation, so it was not used during CSC integration testing.

4.5.5.1 Preparing and Executing the Tests

Figures 4.5 and 4.6 illustrate the overall flow of test case preparation and execution for each of the testing techniques. Test driver development was treated as dynamic test setup activity and was not an integral part of the preparation of the tests themselves.

4.5.5.1.1 Test Preparation

Preparing for test execution included test data preparation and formulation of expected results. Test data preparation formulates test cases and the data to be input to the program. Test case preparation was not applicable to the static testing techniques. For the dynamic testing techniques, test preparation was accomplished through both manual and automated methods.

Test cases were chosen as the result of analyzing the requirements and design specifications and the code itself. Test data was prepared to demonstrate and exercise externally visible functions, program structures, data structures, and internal functions. Each test case included a set of input data and the expected results. The expected results were expressed in terms of final values or intermediate states of program execution.

Testers developed test cases, test data and expected results through examining the program specifications (in particular, the design and program code) according to the procedure of a particular testing technique. Test cases had the objective of demonstrating that the functions, interface requirements, and solution constraints are satisfied according to the objectives of a given testing technique. Test cases were determined from the inputs, functions and structures of the design and code. Test data were determined from the program to exercise computational structures implemented within the program code.

TESTERS AND THE TEST CONFIGURATIONS

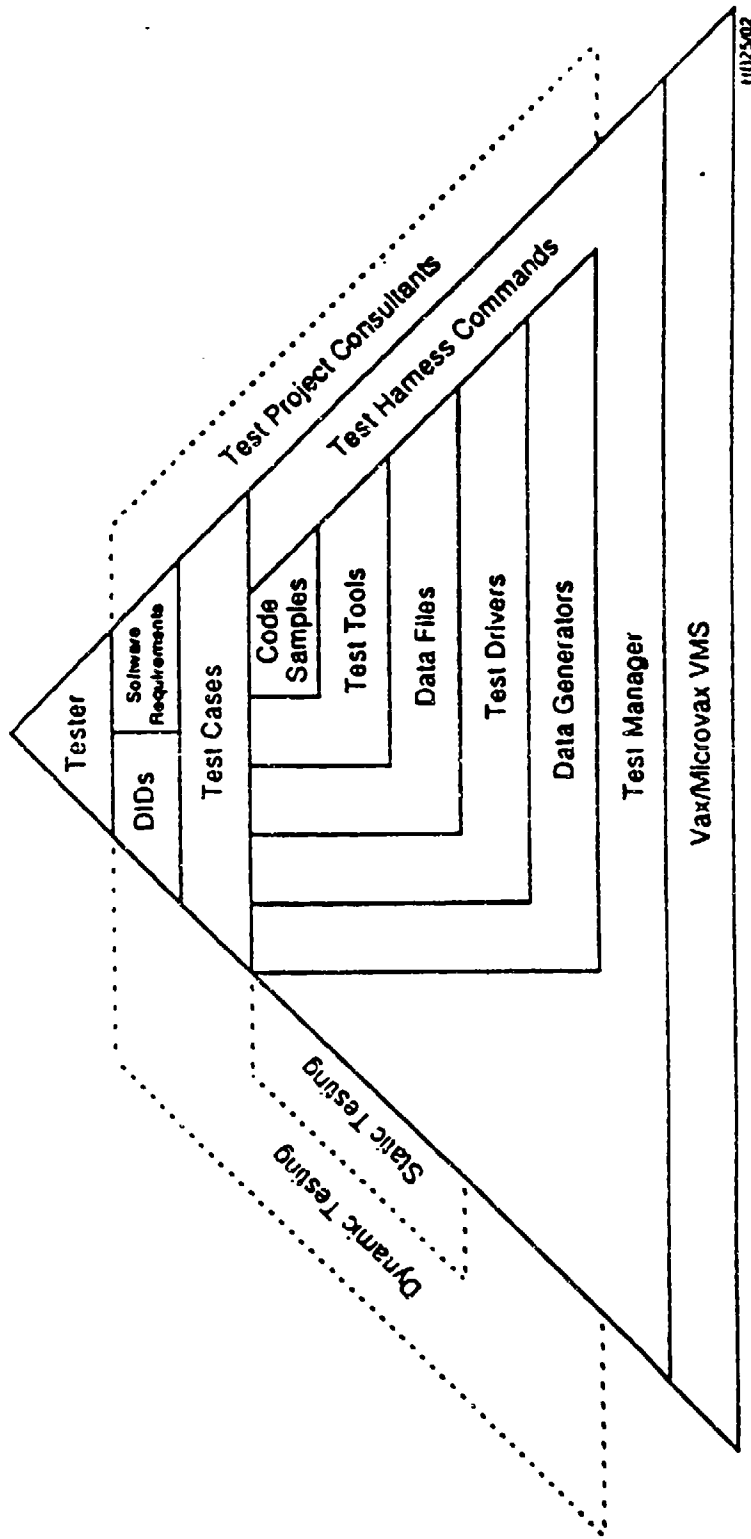


Figure 4.4. Testers and the Test Configurations

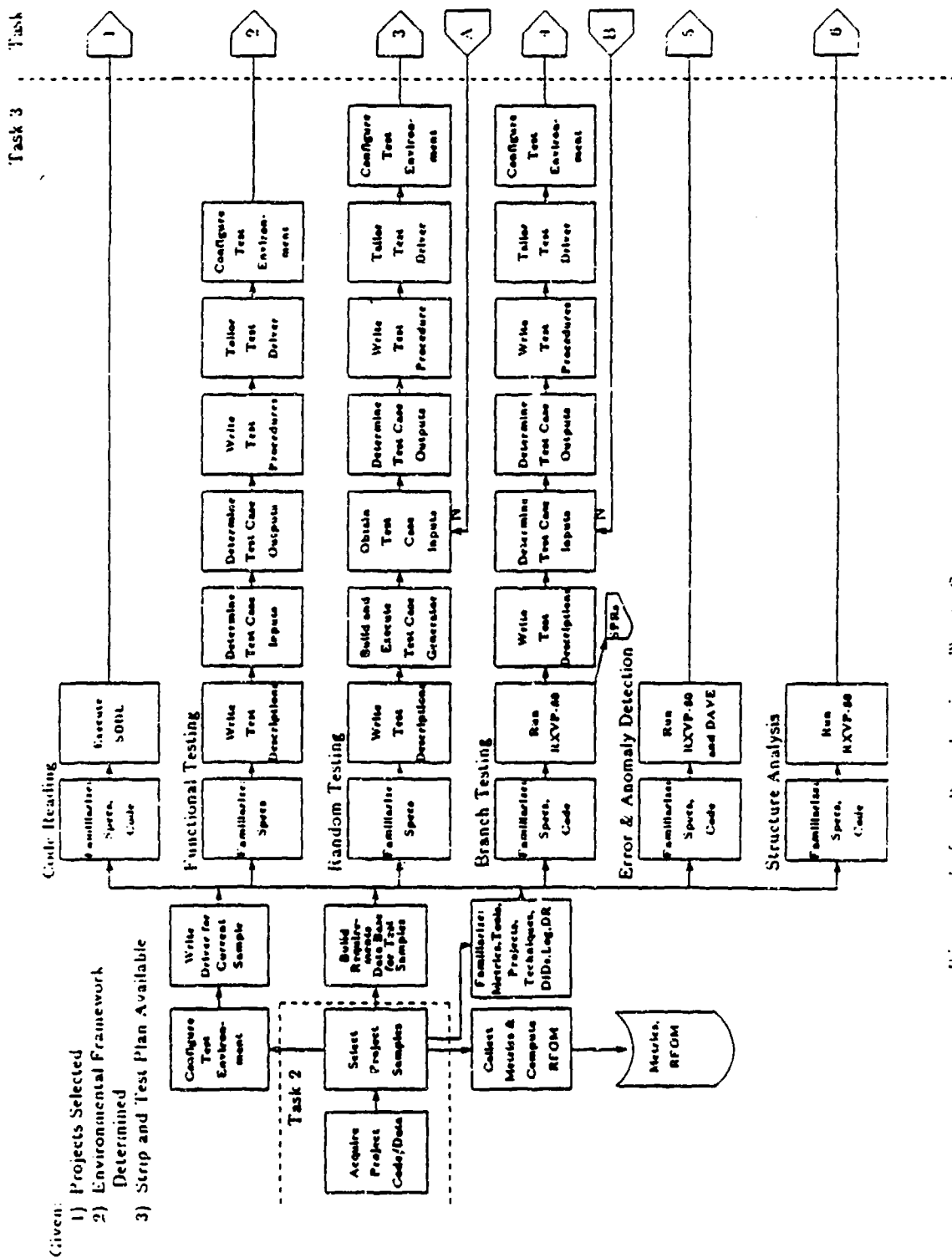


Figure 4.5. Developing Test Cases

Task 4

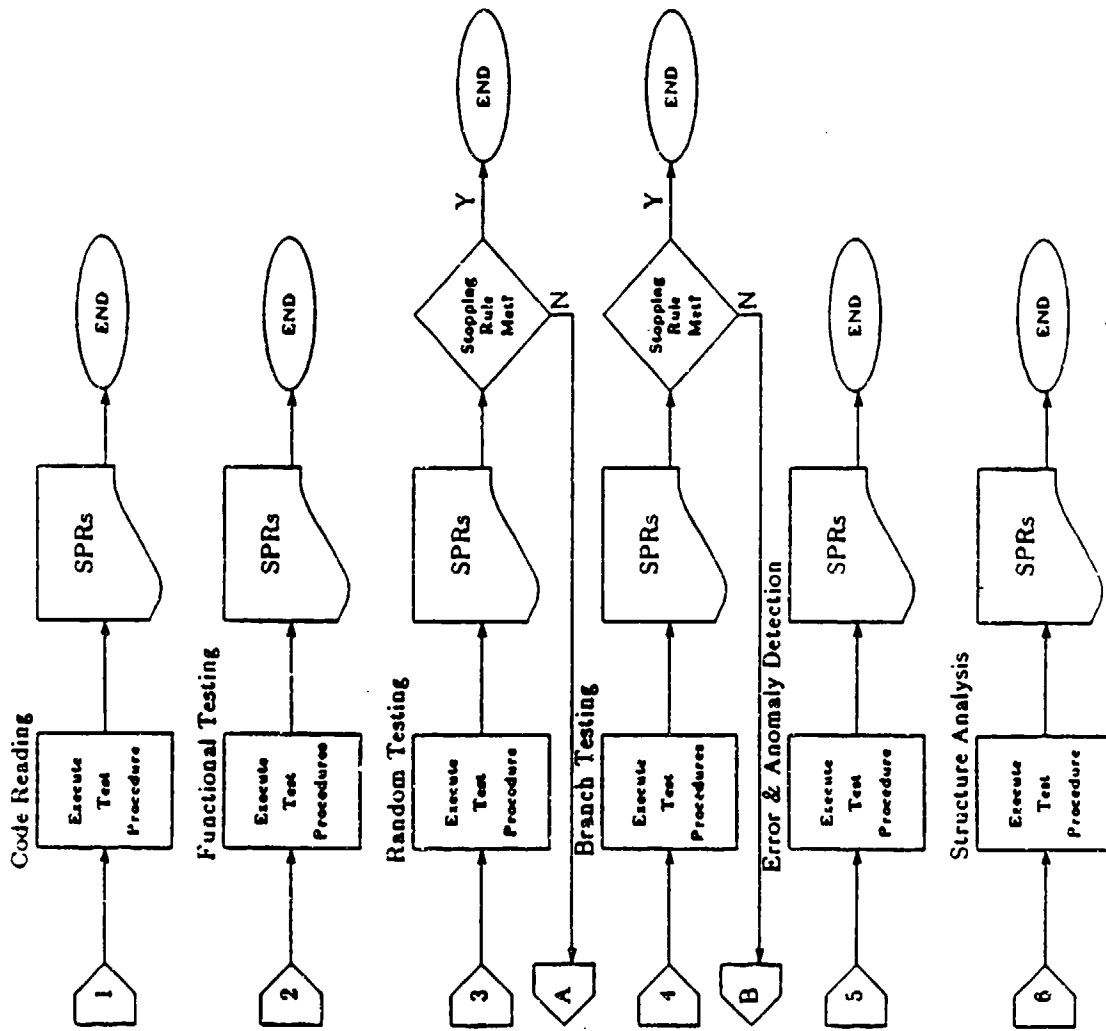


Figure 4.6. Executing Test Cases

The final step before test execution was to tailor the test driver (if needed) to suit any particular needs of the test cases for a given test sample and testing technique.

4.5.5.1.2 Test Execution

For the dynamic techniques, test execution involved executing a program with prepared test cases and then collecting the results. For the static techniques, test execution involved the execution of RXVP-80 or SDDL, as appropriate, and the evaluation of the applicable hardcopy outputs according to the procedures of the given technique.

Dynamic testing was performed in a bottom-up fashion. Bottom-up testing consisted of testing individual units and small collections of units in that order, as though they were not yet integrated into the total program. This required the use of test drivers and stubs or dummy routines for interfacing units not under test.

4.5.5.1.3 Stopping Rules

Stopping rules are not well defined, in general, for testing techniques. The existing options include the following: wall clock time, reaching a specified mean time to failure (MTTF), number of errors found, and exhausting the test technique. An important consideration in choosing stopping rules for the techniques was the desire to compare technique effectiveness and effort across techniques.

The stopping rules shown in Table 4.12 were carefully devised as the best determinable to support these experiment goals. They are derived from the 'Revised Estimates' data in Table 4.13. These estimates which were projected during Task 2 are the total number of hours from Tables 4.14 and 4.15 to complete each step of a given testing technique.

Due to effort budget constraints, it became evident as the experiment conduct progressed that we would have to restrict the amount of effort available to CSC integration level testing. This was accomplished by adopting the unit test level stopping rules for CSC integration testing. The results of this reduced effort are assessed in Chapter 5.

4.5.5.2 Test Analysis

Test analysis was performed by the testers for each dynamic testing technique to capture and report execution details (e.g., branch execution counts) and to determine the thoroughness of the testing. For the static testing techniques, analyses is an integral part of their execution as described in Section 4.5.5.1.2.

The process of analyzing the dynamic test results included comparing the actual to expected outputs. This analysis required a specification of the expected outputs for each test case. Since the output data for all non-interactive tests was machine readable, an automated comparator was used. Interactive outputs were evaluated visually while the tests ran. Upon completion of the test outputs analyses for all dynamic tests for a given code sample, further analyses of test coverage were made and recorded on the Test Coverage Summary Worksheet for all execution paths, inputs, units, interfaces and requirements, as applicable to the test level.

The concluding step was for each tester to evaluate the static and dynamic techniques to determine the unique error found, both individually and by more than one

Table 4.12. Stopping Rules

TEST TECHNIQUE	TEST PERSONNEL	STOPPING RULE	TEST LEVEL	
			UNIT	CSC*
Branch Testing	Tester	Branches identified through static analysis, test descriptions & procedures complete, test case outputs determined, & on line environment configured. Not to exceed X hours.	X = 21	X = 42
	Operator	100% of branches executed (with a minimum of 2 traversals per branch) and MTTF = 10 input cases. Not to exceed X hours.	X = 8	X = 16
Code Review	Tester	All required aspects of the method have been evaluated using SDDL where possible, manually where not. Not to exceed X hours	X = 8	X = 16
Functional Testing	Tester	Driver in place and online environment configured. Not to exceed X hours.	X = 12	X = 24
	Operator	All test procedures executed. Not to exceed X hours.	X = 4	X = 8
Random Testing	Tester	Test descriptions & procedures written, test driver & test case generator ready, test case outputs determined, online environment configured. Not exceed to X hours.	X = 16	X = 32
	Operator	Minimum number, Y, samples from input space executed, and MTTF = 10 input cases. Not to exceed X hours.	X = 6 Y = 25	X = 12 Y = 50
Error & Anomaly Detection	Tester	All required aspects of the method have been evaluated, using automated tool where possible, manually where not. Not to exceed X hours.	X = 6	X = 12
Structure Analysis	Tester	All required aspects of the method have been evaluated, using automated tool where possible, manually where not. Not to exceed X hours.	X = 4	X = 8

* Unit test level stopping rules utilized for CSC Integration Test Level due to effort budget constraints.

Table 4.13. Test Effort Estimates (Hours)

TEST LEVEL	TEST TECHNIQUE	ORIGINAL ESTIMATE				REVISED ESTIMATE			
		TASK 3	TASK 4	1 TESTER	4 TESTERS	TASK 3	TASK 4	1 TESTER	4 TESTERS
UNIT TEST	C	27	5	32	128	0	8	8	32
	F	21	1	22	88	12	4	16	64
	R	40	4	44	176	16	6	22	88
	B	14	4	18	72	21	8	29	116
	E	20	5	25	25	0	6	6	6
	S	20	2.5	22.5	22.5	0	4	4	4
(Totals)	1 sample/tester	142	21.5	163.5	511.5	49	36	85	310
	2 samples/tester	284	43	327	1023	98	72	170	620
CSC TEST	C	0	12	12	48	0	16	16	64
	F	21	2	23	92	24	8	32	128
	R	40	16	56	224	32	12	44	176
	B	3	18	70	280	42	16	58	232
	E	0	6	6	6	0	12	12	12
	S	0	6	6	6	0	8	8	8
(Totals)	1 sample/tester	113	60	173	656	98	72	170	620
	2 samples/tester	226	120	346	1312	196	144	340	1240
CSCI TEST	F	24	3	27	108	48	16	64	256
	R	44	40	84	336	64	24	88	352
(Totals)	1 sample/tester	68	43	111	444	112	40	152	608
	2 samples/tester	136	86	222	888	224	80	304	1216
Total estimated hours					3223				3076
Max. available hour**					3200				3200

*Note: Only 1 tester for Error and Anomaly detection (E) and Structure Analysis (S)

** Note: 4 test full-time for 5 months (Jan through May, 1988)

C-Code Review
F-Functional Testing
R-Random Testing

B-Branch Testing
E-Error and Anomaly
S-Structure Analysis

Table 4.14. Unit Test Effort Worksheet A: Driver & Static Techniques

Technique	Time (Hr)	Step
Driver Development	2	familiarization with code and specifications**
	2	document driver requirements**
	4	document driver design
	4	develop and test driver**
	12	All steps
Code Review	1	familiarization with materials**
	1	run SDDL**
	4	review code and documentation with checklist***
	2	write SPRs***
	8	All steps
Error and Anomaly Detection	1	familiarization with materials**
	1	run RXVP-80**
	2	review tool results***
	2	write SPRs***
	6	All steps
Structure Analysis	1	familiarization with materials**
	1	run RXVP-80**
	1	Review tool results***
	1	enter log and write SPRs***
	4	All steps

* Prorated, as 4 hours were added to the Task 3 portion of each dynamic test technique.

** Designates Task 3 Activities.

*** Designates Task 4 Activities.

Table 4.15. Unit Test Effort Worksheet B: Dynamic Techniques

Technique	Time (Hr)	Step
Functional Testing	1	familiarization with materials**
	2	write inputs to test description**
	2	write inputs to test procedure**
	2	adapt test driver**
	1	set up test harness**
	1	execute tests***
	1	review results***
	2	write SPRs***
	12	All steps
Random Testing	1	familiarization with materials**
	2	determine random data (specify generator)**
	2	write inputs to test description**
	2	write inputs to test procedure**
	2	develop I/O data (write generator)**
	2	adapt test driver**
	1	set up test harness**
	2	execute tests***
	2	review results***
	2	write SPRs***
	18	All steps
Branch Testing	1	familiarization with materials**
	3	determine branch data**
	3	write inputs to test description**
	3	write inputs to test procedure**
	3	develop I/O data**
	3	adapt test driver**
	1	set up test harness**
	3	execute tests***
	3	review results***
	2	write SPRs***
	25	All steps

technique, and whether each error found was a known development error or newly detected during the experiment. This information was recoded on the Error Summary Worksheet, with the ID and description of each error and the total number of errors for each technique.

4.5.5.3 Data Collection and Organization

Collection of the preceding data was designed and conducted with the goal of providing valid quantitative data that can be meaningfully analyzed, interpreted, and referenced during the writing of an integrated software reliability measurement and test technique guidebook. The analysis of these data will provide insight into selecting software testing techniques, determining how much test effort should be planned, and making decisions concerning the level of software reliability attained.

The collected raw test technique data is in the Task 4 Report. The following sections contain a discussion of the data collection and organization methodology utilizing worksheets, the 4th Dimension DBMs and StatView data analysis tools.

4.5.5.3.1 Data Collection

Careful thought was given to the data to be collected and to the data collection procedures. General forms and procedures that support the test technique data collection activities are described in the Software Test Plan. They are derived in part from DoD-STD-2167 for Test Description, Test Procedure and Test Report; from the SRPEG metric data collection forms for the REN; and from statistical data analysis requirements related to discrepancy reporting, execution time, failure rate, test effort, test coverage and test methodology. The resulting test data collection procedures and forms for the SRMTIT experiment are provided in Volume 2 of this report.

Thirteen data collection worksheets were developed for use. Table 4.16 identifies these worksheets by name and ID, lists who was responsible for completing them, and lists the activities for which they were used. Note that Worksheets T2 and T12 were not needed and were dropped from the experiment. Table 4.17 shows a correlation between the test worksheets and their data collection counterparts in the SRPEG. All of the information on these worksheets was entered into the Test Database as indicated in Figure 4.16 (except for the final text field on worksheets T1, T4, T6 and T8, due to an anomaly in the DBMS. All of these worksheets except T13 are included in the applicable Test Description, Test Procedure and Test Report documents in their entirety.

Worksheet T13, Experiment Error Summary, is included in Chapter 5. All detected errors of each test technique for each code sample were summarized into tables as a means of gathering this information to input into the StatView database. All detected errors were correlated with the original development SPRs and entered into worksheet T13.

4.5.5.3.2 Data Organization

Figures 4.7 and 4.8 provide an organizational overview of the data that was collected as a part of this effort. These figures depict the data structures around which the data collection effort is organized. Figure 4.7 shows the 4th Dimension raw test data files. Figure 4.8 shows the tabulated StatView data files. These logical organizations of data were chosen for the following four reasons:

Table 4.16. Test Worksheet Summary

Worksheet ID	Worksheet Name	Responsible User	Applicability							Used	Auto
			B	F	R	C	E	S	G		
T1	Test Activity Log	T, C	X	X	X	X	X	X		Y	4, S
T2	Test Configuration & Data Eval.	T							X	N	
T3	Software Requirements List	A	X	X	X					Y	
T4	Test Name & Objectives	T	X	X	X					Y	4, S
T5	Test Case Description	T	X	X	X					Y	
T6	Test Procedure Specification	T	X	X	X					Y	4, S
T7	Test Execution Summary	T, C	X	X	X					Y	4, S
T8	Test Execution Log	T	X	X	X					Y	4, S
T9	Software Problem Report	T	X	X	X	X	X	X		Y	4, S
T10	Test Coverage Summary	T	X	X	X					Y	4, S
T11	Test Technique Selection	T							X	Y	4, S
T12	Test Problem Summary	T	X	X	X	X	X	X		N	
T13	Experiment Error Summary	T	X	X	X	X	X	X		Y	S

Responsible User:

A: Analyst

T: Tester

C: Consultant

Used:

Y: Yes

N: No

Automation:

4: 4th Dimension

S: Statview

Applicability:

B: Branch Testing

R: Random Testing

E: Error & Anomaly Detection

F: Functional Testing

C: Code Review

S: Structure Analysis

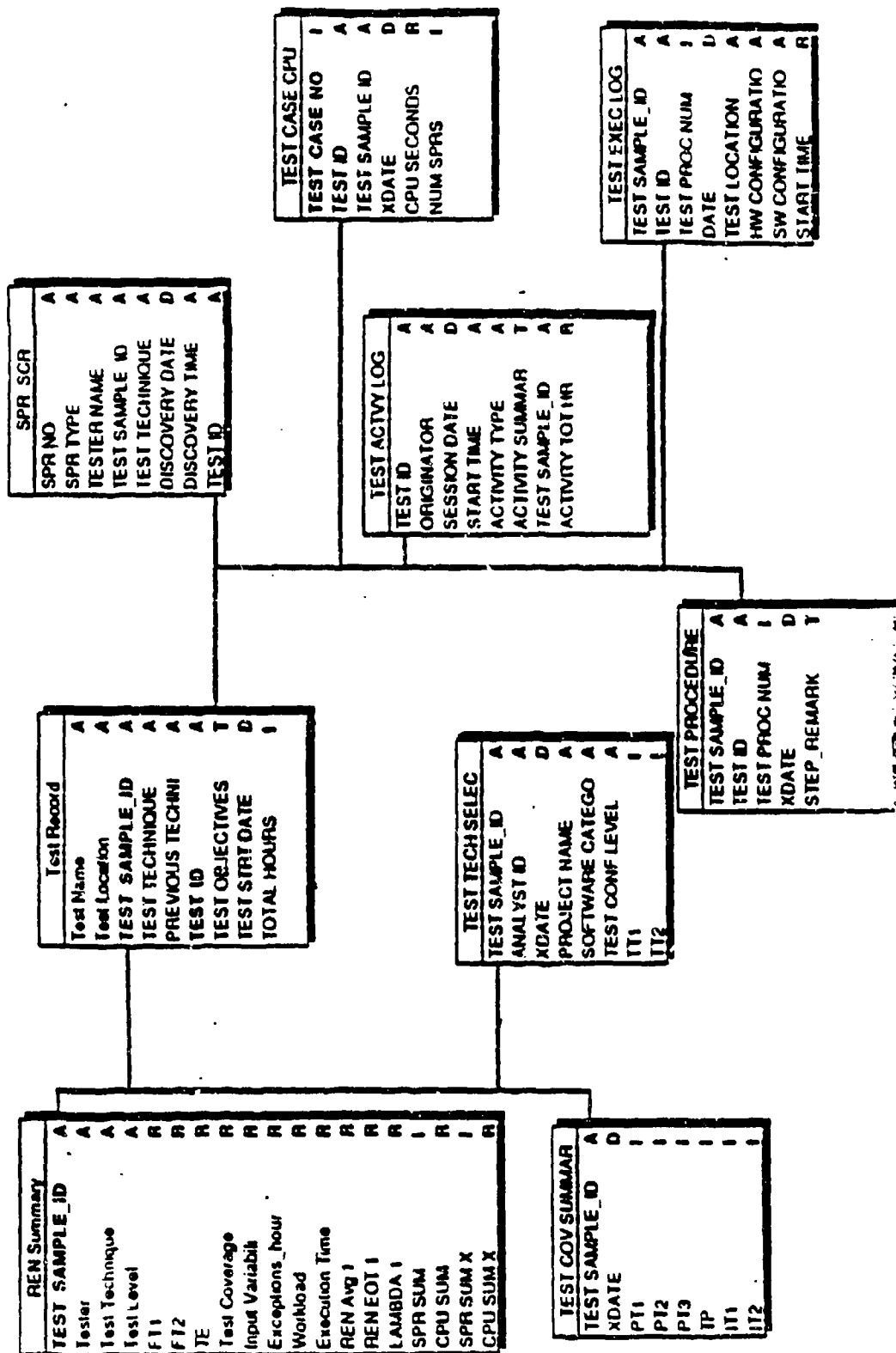
G: General Use

Table 4.17. Cross-reference of Test Data Worksheets and SRPEG Metric Worksheets and Procedures.

METRIC DATA (TASK 201)	SRPEG DATA COLLECTION PROCEDURE	SRPEG METRIC WORKSHEETS	SRMTIT TEST DATA WORKSHEETS
Average Failure Rate During Test (FT1)	12, 13, 14	5, 6	T7, T8, T9
Failure Rate at End of Test (FT2)	12, 13, 14	5, 6	T7, T8, T9
Test Effort (TE)	15	6	T1
Test Method (TM)	16	7	T11
Test Coverage (TC)	17	8	T10

Figure 4.7 DBMS RAW TEST DATA FILES

Monday, April 4, 1988 Structure for REN DATABASE



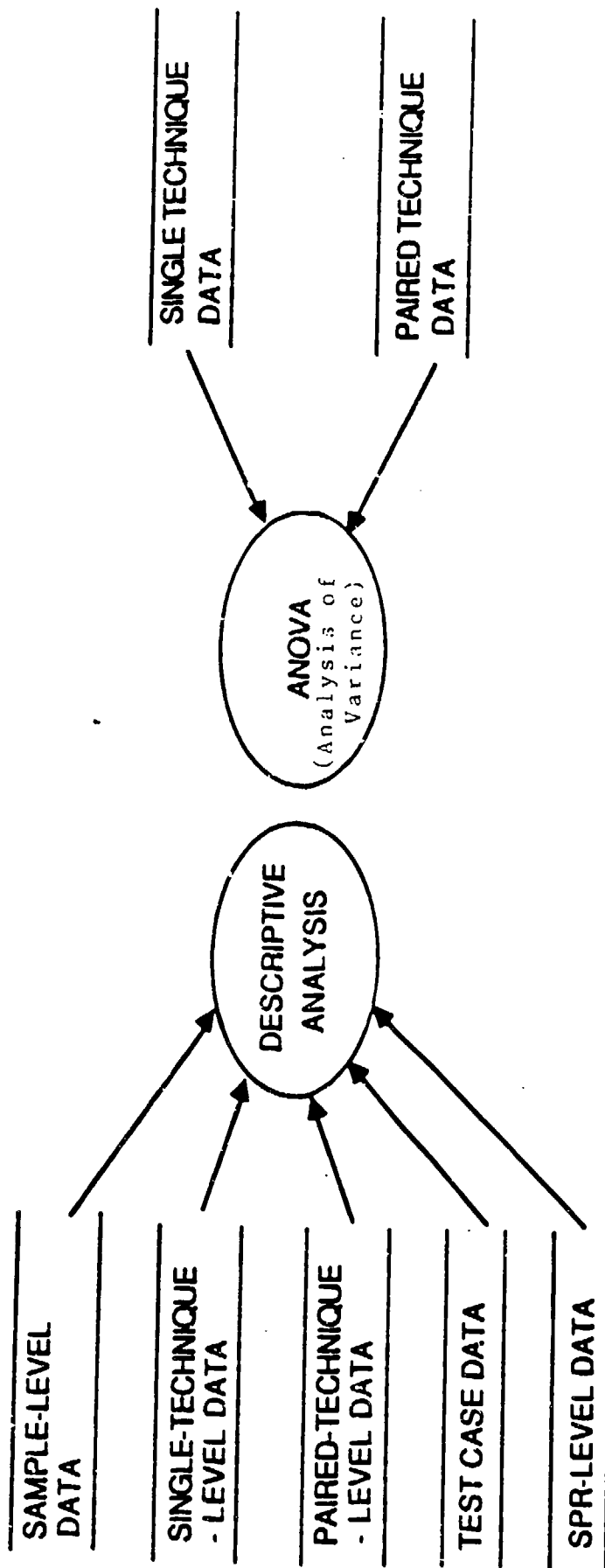


FIGURE 4.8 STATVIEW DATA FILES.

- a. To parallel the distinction between experiment activities (i.e., reliability measurement data collection vs. test technique evaluation).
- b. To permit easy file update associated with these activities by multiple personnel at multiple sites.
- c. To facilitate data selection for combined analysis.
- d. To provide a foundation for reuse of the reliability measurement and test technique data for future experiments.

All raw test results data recorded on the worksheets by each tester were entered manually into the Test Database. From here these data were converted into StatView data files for detail analyses.

The test data files are organized by type of test worksheet and are provided separately in the Task 4 report, in the companion 2167A test documents, and in electronic form compatible with the IRMS.

Data tabulated in StatView to support analysis are derived from the RPFOM database (i.e., Lines of Code and Complexity), and nonautomated test worksheets. They are provided separately in the Task 4 report and in electronic form, and appear in tables identified by corresponding StatView file names. A detailed requirements specification for the StatView database also is provided in the Task 4 report. These requirements specify seven data files organized by level of analysis, level of detail, and level of software component.

Five StatView files are designated for descriptive analyses (see Figure 4.8) of Unit and CSC data at the following levels of detail: test sample, test technique (single and paired), test case, and software problem report (SPR). The remaining two StatView files are designated for Analysis of Variance (ANOVA) of unit data at the test technique level. One file is specified for data on single techniques, and the other file is specified for data on paired techniques.

4.6 REN Results

A Reliability Estimation Number (RENAVG) was computed for each test sample unit and CSC (i.e., "integrated unit") based upon average observed failure rate during the software testing experiment. Computation of RENEOT, based on end-of-test failure rate, was considered infeasible due to the absence of regression testing and the difficulty in defining a "test period."

Formal testing is usually accompanied by "regression testing" in which software components are tested, debugged, and then tested again in an iterative fashion. This technique, which tends to minimize incidences of recurring errors and thus lead to improved reliability as testing progresses, was not employed during the experiment. Consequently, recurring errors were often prevalent.

The closest analogy to "test period" in the present study is "test case." Due to the small time intervals involved, tracking of computer operation time for individual test cases was not feasible. Although CPU time was measured for test cases, values were generally negligible and often unmeasurable (i.e., equal to zero).

Table 4.19

AVERAGE OBSERVED FAILURE RATE DURING TESTING (F_{T1})

Test Samples	Experiment SPRs	Dynamic Technique SPRs	Dynamic Technique Test SPRs	Dynamic Technique Test Time (hrs.)	F_{T1}
UASNWMSN	4	3	2	1.42	1.41
UATHUADC	2	0	0	2.17	0
UATHUSCN	0	0	0	0.92	0
UATHXRDT	13	9	8	1.75	4.57
USINFANSEN	7	6	4	4.34	0.92
USINMODE	20	15	13	3.94	3.30
USINPSAT	12	8	6	10.86	0.55
USSCANNER	17	1	0	3.75	0
CASNWMSN	5	5	5	0.70	7.14
CATHPCON	16	15	11	1.00	11.00
CATHUADS	2	1	1	0.53	1.89
CATHURDT	3	3	0	0.31	0

F_{T1} = Dynamic Technique Test SPRs/Dynamic Technique Test Time
 = Errors per hour

Determination of the REN component metrics values - FT₁, TE, TM, and TC - is detailed below, followed by the REN calculations.

4.6.1 Average Test Failure Rate (FT₁)

The average failure rate observed during testing, which may be determined at any point in the testing process, serves as a baseline for the REN_{AVG} calculation:

$$FT_1 = \text{Total no. SPRs during testing} / \text{total test time}$$

The denominator, total test time, may be measured in computer operation time or CPU time. Computer operation time, measured in hours, was selected for the test sample RENs. This unit of measurement is considered by the SRPEG to be preferable to CPU time. In addition, as noted above, CPU times recorded for experimental testing were usually negligible due to the small sizes of the unit and "integrated unit" (i.e., CSC) test samples.

Since computer operation time is unavailable for the static test techniques, FT₁ reflects dynamic testing only. The test sample SPR counts and test times for dynamic testing are presented in Table 4.19. The Dynamic Technique SPRs represent a subset of the total Experiment SPRs generated for both dynamic and static testing. The Dynamic Technique Test SPRs, in turn, represent the subset of Dynamic Technique SPRs logged during actual test conduct and during comparison of actual test output to expected test output.

The Dynamic Technique Test Time values represent computer operation times recorded by testers for dynamic testing. The relatively high values logged for the four SCENE unit test samples reflect interactive processing of tester inputs, which contrast with the batch processing of inputs for the AFATDS unit and CSC test samples. The relatively low values recorded for AFATDS CSC test samples probably reflect in part the discontinuation at CSC testing of production of benchmark files containing expected output values, and the possibility of fewer test cases submitted compared to AFATDS unit testing.

The average test failure rates (FT₁) in Table 4.19 are expressed as errors (SPRs) per hour. At least one test sample from each of AFATDS unit testing, SCENE unit testing, and AFATDS CSC testing experienced a zero failure rate due to the absence of SPRs logged during dynamic testing. Failure rates for AFATDS CSCs appear relatively high compared to those of the other test samples. Empirically, this seems due to the lower test times for the CSCs, rather than to a greater number of SPRs.

4.6.2 Test Effort (TE)

Test effort is based on percent of development effort devoted to testing, which is unavailable for the SCENE and AFATDS projects. Consequently, the assumption is made that a minimum of 40% of development effort was devoted to testing, resulting in a test effort of 1.0.

4.6.3 Test Methodology (TM)

The Test Methodology metric is assigned one of three values (i.e., 0.9, 1.0, or 1.1) based on the proportion of testing techniques actually employed (TU) that are recommended (TT) by one of three technique selection paths in the Software Test Handbook (STH). The higher the proportion of techniques used, the lower the value

Table 4.20

TEST METHODOLOGY (TM)

Test Samples	TU/TT(1)	TU/TT(2)	TU/TT(3)	TM(1)	TM(2)	TM(3)
UASNWMSN	0.33	0.38	0.38	1.1	1.1	1.1
UATHUADC	0.56	0.54	0.54	1.0	1.0	1.0
UATHUSCN	0.33	0.38	0.38	1.1	1.1	1.1
UATHXRDT	0.56	0.54	0.54	1.0	1.0	1.0
USINFANSEN	0.33	0.38	0.38	1.1	1.1	1.1
USINMODE	0.33	0.38	0.38	1.1	1.1	1.1
USINPSAT	0.56	0.54	0.46	1.0	1.0	1.1
USSCANNER	0.56	0.54	0.46	1.0	1.0	1.1
CASNWMSN	0.56	0.54	0.54	1.0	1.0	1.0
CATHPCON	0.56	0.54	0.54	1.0	1.0	1.0
CATHUADS	0.33	0.38	0.38	1.1	1.1	1.1
CATHURDT	0.33	0.38	0.38	1.1	1.1	1.1

TM(1) --> STH Path 1 based on Software Category
 TM(2) --> STH Path 2 based on Test Phase and Objectives
 TM(3) --> STH Path 3 based on Software Error Categories

of the TM multiplier, resulting in a lowering of the estimated failure rate (REN). The TU/TT proportions and corresponding TM values for test samples appear in Table 4.20.

Differences in TU/TT among test samples for a particular technique selection path (e.g., TU/TT(1) in Table 4.20) reflect differential utilization of two static test techniques, Error and Anomaly Detection (EA) and Structure Analysis (SA). Thus, test sample UASNWMSN, which was not tested using EA or SA techniques, exhibits lower TU/TT values across technique selection paths than UATHUADC, for which EA and SA testing were employed.

Differences in TU/TT across technique selection paths for a particular test sample (e.g., values of 0.56, 0.54, and 0.46 for USINPSAT) reflect variation in number of techniques recommended. In the case of paths 2 and 3, $TT = 13$, which explains the equivalency of values observed for most test samples. The difference in values for TU/TT(2) and TU/TT(3) for USINPSAT and USSCANNER is due to the omission of SA from recommendations in path 3; this technique is recommended in path 2, and is employed in testing of both test samples.

The *a priori* selection of test techniques, as occurred in this study, poses an interesting dilemma with regard to utilization of the STH test technique selection procedures. In some instances, techniques employed (TU) are not among those recommended (TT). In computing the values in Table 4.20, techniques that were not recommended by a technique selection path but that were nevertheless applied in testing a sample were excluded from determination of TU for that sample within that selection path, resulting in relatively lower TU/TT values and potentially higher TM values. Although consistent with the STH selection procedures, this approach ignores the possible contribution to greater software reliability of application of testing techniques excluded from the recommendations.

The alternative approach of including as viable components of TU the non-recommended techniques employed in testing can take two directions. The recommended and non-recommended techniques can be weighted equally (i.e., in either case, TU is incremented by one for each technique added), or the latter can be weighted less (e.g., TU is incremented by 0.5 for each non-recommended technique added). Either method can potentially result in a TU/TT value exceeding 1.0, but this does not violate the existing model for determining TM. Fractional weighting of non-recommended techniques is attractive since, while this approach recognizes the potential contribution to software reliability of any addition to testing methods employed, it takes into consideration the relatively greater contribution expected from methods recommended from technique selection analysis.

4.6.4 Test Coverage (TC)

Three procedures are available for computing Test Coverage. Selection of procedure depends primarily on the subject software component level, which determines the data available for the calculation. In each approach, TC represents the inverse of the extent of test coverage. Thus, as test coverage declines, TC increases, resulting in a higher estimated failure rate. Computation of Test Coverage for the unit and "CSC" (i.e., integrated units) test samples is based on proportions of total execution paths (measured using RXVP-80) and total inputs that were tested.

Results of Test Coverage determinations for test samples are presented in Table 4.21. Complete test coverage ($TC = 1.0$) is evident only in the three test samples which possess the smallest number of execution paths. Differences in TC for the remaining

test samples is entirely due to variation in proportion of total execution paths tested, since all inputs were successfully tested for all test samples.

4.6.5 REN Calculation

Values of REN_{AVG} for test samples appear in Table 4.22. Average test failure rates (FT_1) are listed for comparison. The most striking observation of these results is the drastic reduction in average test failure rates represented by the estimated failure rates. Given the values derived for TE, TM, and TC, the principle factor contributing to the differences in values between these failure rates is the 0.02 coefficient. This constant represents an attempt to adjust for the inherent tendency of software testing environments to increase the potential for detecting errors. Specifically, this coefficient is derived from empirical evidence that suggests average test failure rates exceed operational failure rates by 50 times. Aside from the FT_1 baseline values, the operational failure rate coefficient is by far the primary determinant of the REN_{AVG} values for the test samples relative to the other metric multipliers.

Mean REN_{AVG} values for AFATDS unit samples, SCENE unit samples, and AFATDS CSC samples are given in the last column of Table 4.22. Similar means, which appear equivalent after rounding off, for AFATDS and SCENE unit samples suggest that the nature of the testers' interaction with the system (batch for AFATDS, interactive for SCENE) does not influence estimated failure rate. The relatively larger mean REN_{AVG} values for CSC samples are undoubtedly a consequence of lower test times (Table 4.19), which resulted from cessation of benchmark file generation during CSC testing. Consequently, these data probably do not effectively demonstrate a difference between unit and CSC test samples in average estimated failure rate.

4.7 Tester Profiles/Feedback

Tester Profile questionnaires were completed by each tester prior to the conduct of the software testing experiment. The questionnaire summarized the education and experience of each tester prior to the experiment. These results are presented in Chapter 5.

A survey was also performed upon completion of the pilot experiment. Each participating tester was queried informally for his/her opinions regarding the test techniques and tools, test environment and process, and the test worksheets. They were asked to comment on these topics based on their own experience as gained during the pilot. The results of this survey are contained in the Task 4 Report [14] and are included in discussions in Section 6.2.5 and 6.2.6.

Table 4.21

TEST COVERAGE

	SAMPLE	PT	TP	IT	TI	PT/TP	IT/TI	TC
1	UASNWMSN	8	8	13	13	1.000	1.000	1.000
2	UATHUADC	8	8	7	7	1.000	1.000	1.000
3	UATHUSCN	9	9	3	3	1.000	1.000	1.000
4	UATHHRDT	27	31	3	3	.871	1.000	1.069
5	USINFANSEN	97	108	10	10	.898	1.000	1.054
6	USINMODE	93	106	27	27	.877	1.000	1.065
7	USINPSAT	134	155	28	28	.865	1.000	1.073
8	USSCANNER	77	79	3	3	.975	1.000	1.013
9	CASNWMSN	20	22	20	20	.909	1.000	1.048
10	CATHPCON	39	49	5	5	.796	1.000	1.114
11	CATHUADS	19	24	9	9	.792	1.000	1.116
12	CATHURDT	46	82	1	1	.561	1.000	1.281

Table 4.22. REN Calculations

Test Sample	F_{T1}	REN AVG	$\overline{\text{REN}}$ AVG
UASNWMSN	1.41	0.03	0.03
UATHUADC	0	0	
UATHUSCN	0	0	
UATHXRDT	4.57	0.10	
USINFANSEN	0.92	0.02	0.03
USINMODE	3.30	0.08	
USINPSAT	0.55	0.01	
USSCANNER	0	0	
CASNWMSN	7.14	0.15	0.11
CATHPCON	11.00	0.25	
CATHUADS	1.89	0.05	
CATHURDT	0	0	

$$\text{REN}_{\text{AVG}} = (F_{T1}) 0.02 (TE \cdot TM \cdot TC)$$

$$TE = 1.0$$

5.0 EXPERIMENT FINDINGS

This chapter reports upon the work undertaken to analyze the experiment's results. Analyses were performed on the test data from six experimentally controlled testing techniques and on the reliability prediction methodology.

5.1 Scope of Analysis

Analyses completed on the test data include descriptive and more formal analyses. Descriptive analyses consist of bar charts and tables which present the raw and reduced data in visual form. Bar charts are provided that describe static properties of the code samples such as lines of code and complexity. For example, tables are provided which show percent of Software Problem Reports (SPRs) found by technique and percent of SPRs found by each pair of techniques. Descriptive analyses were done at both the unit and Computer System Component (CSC) levels.

Formal statistical analyses were performed at the unit level. At that level sufficient data points were obtained to justify such analyses. Analyses of variance (ANOVAs) were conducted to test whether differences exist among test techniques in 1) their effectiveness in finding errors, 2) their branch coverage performance, 3) the effort needed to apply them, and 4) the efficiency with which they uncovered errors.

The data used in the analyses are stored in two formats: 1) in online files and on diskette, for use with the Macintosh software package StatView 512+ and 2) in the error summary tables completed by the testers, and validated by project consultants (the tables appear in Section 5.2.6). There appear to be minor discrepancies across these two data repositories; the contract completion schedule precluded resolving those discrepancies, so we suggest a data validation across the two repositories as a starting point for future work with these data. The inconsistencies appear minor, so it is likely analysis results contained herein are correct or very nearly so.

5.2 Analyses of the Test Data

5.2.1 Descriptive Analyses

These analyses provide a summary description of the samples, Software Problem Reports (SPRs) logged against the samples, and test technique performance overviews. The descriptive analysis of the test technique data involves the construction of tables and frequency histograms described in Appendix V of the Task 2 Report (SRTIP) [6]. Many analyses include the two deterministic techniques: Error and Anomaly Detection and Structure analysis, as well as the four nondeterministic techniques: Branch, Code Review, Functional, and Random testing.

5.2.1.1 Sample Description

Samples were chosen from the two software projects shown in Table 5.1. Additional information describing the projects is available in the Chapter 2.

Table 5-1 Characteristics of Sample Projects

	Contract Type		Program Size (Lines of Source)			Environment				MIL-STD Followed	Metric Data
						Development		Target			
	USAF	DoD	10-50K	50-100K	>100K	HW	SW	HW	SW		
AFATDS (Advanced Field Artillery Tactical Data System)		Y	Y			VAX 11/780	VMS 4.2	VAX 11/750	VMS 4.2	None	
SCENE (Secenario Generator)	Y		Y			VAX	VMS	VAX	VMS	NONE	

At the unit test level, four samples were chosen from each project for use in the latin square experiment design. The four unit samples chosen from the AFATDS project are: SNWMSN, THUADC, THUSCN, and THXRDT. the four unit samples from the SCENE project are: INFANSEN, INMODE, INPSAT, and SCANNER. At the CSC test level, four samples were chosen from the AFATDS project. They are: SNWMSN, THPCON, THUADS, and THURDT. Additional information describing how the samples were chosen can be found in Chapter 4.

The histogram in Figure 5.1 shows lines of code (LOC) for each sample. The LOC measure was taken from AMS tool output, which computed a LOC value equal to the sum of all lines in the files, excluding blank, comment and D (debug) lines. Note that lines of include files were not included in the LOC total for a sample; however, a FORTRAN statement continued onto a second line in the file would count as two LOC.

With this measure, the AFATDS units all exhibited noticeably smaller LOC than the SCENE units which contained from 292 to 680 LOC. The CSC samples from AFATDS are noticeably larger than the unit samples from AFATDS; however, the SCENE units are larger than the AFATDS CSCs. This highlights apparent differences in the design of the two systems. AFATDS was designed to have smaller functional entities as units than was the SCENE system.

The histogram in Figure 5.2 shows sample complexity, as measured by branch complexity, or the sum of unconditional and conditional code branches in each sample. This complexity measure was chosen because it was also specified for use in the Reliability Prediction Figure of Merit (RPFOM) equation. (See Chapter 3 for more information on the RPFOM.)

The complexity histogram takes on the same general shape as the LOC histogram. This shows that the SCENE unit samples are much more complex than the AFATDS units and CSCs. The AFATDS CSCs are only slightly more complex than the AFATDS units.

5.2.1.2 Unit Testing

5.2.1.2.1 Single Test Technique Description

Single Technique Effectiveness

Table 5.2 illustrates the categories that SPRs fall into and how many SPRs fell into each category for each sample. The numbers in the cells of the table represent SPR

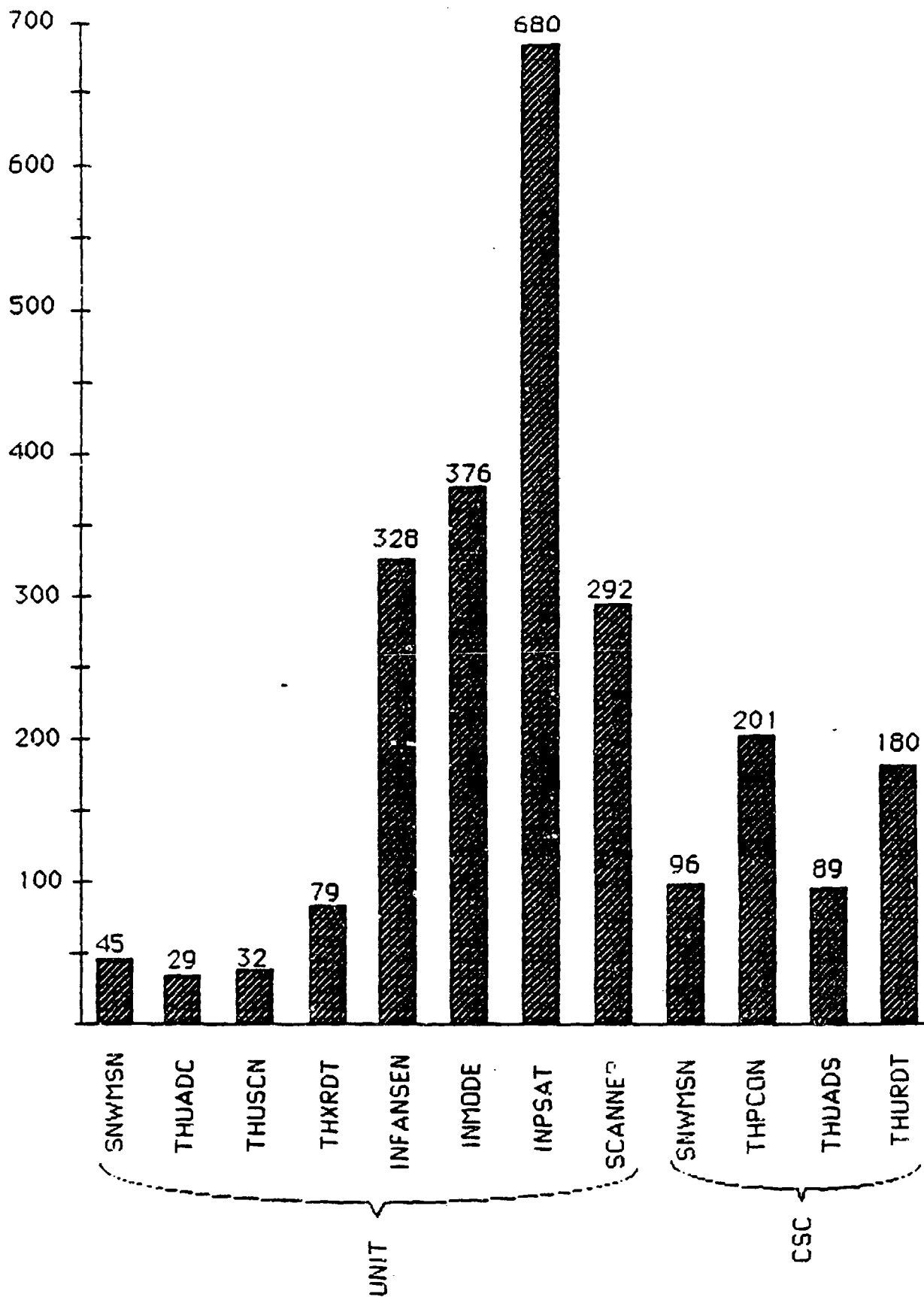


Figure 5.1: Sample Lines of Code

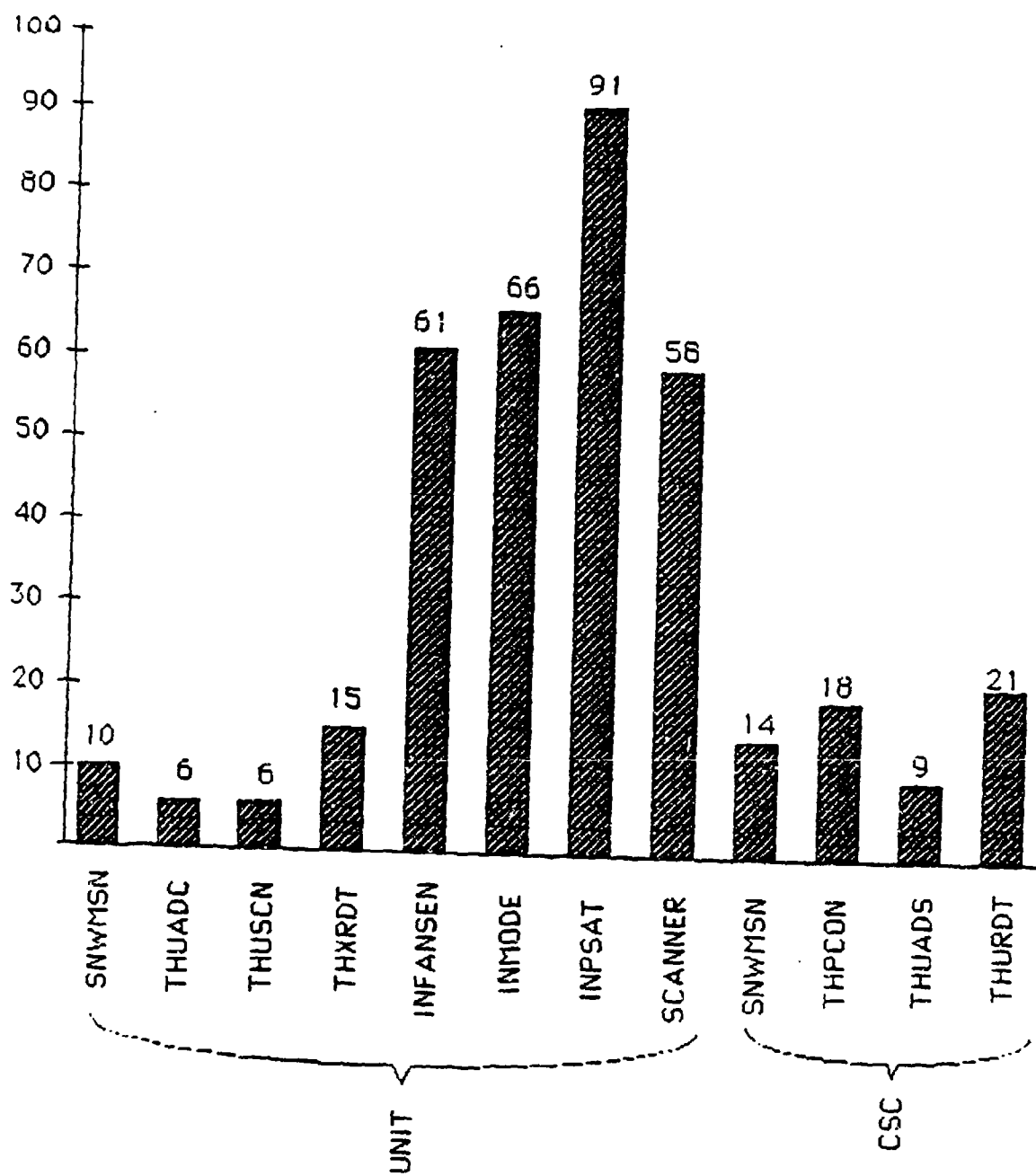


Figure 5.2: Sample Complexity

counts. Each column in this table provides information for the sample named at the heading of the column. Rows are defined as follows:

Table 5.2: Breakdown of SPRs by Unit Sample

UNITS

SPRS	SNWMSN	THUADC	THUSCN	THYRDT	INFANSEN	INMODE	INPSAT	SCANNER
Found In Exper	4	2	0	13	6	20	13	17
Newly Found in Exper	4	0	0	13	6	20	13	16
Orig Findable	0	3	1	1	4	1	1	2
Exper Findable	4	3	1	14	10	21	14	18
% of Exper Findable Found in Exper	100%	67%	0%	93%	60%	95%	93%	94%

Found in Exper: The number of SPRs that were found during the experiment for this sample.

Newly Found in Exper: Whereas the above measure includes any SPRs found in the experiment that had already been found during the original code development, this count represents only those SPRs found during the experiment runs that reported new errors not previously found during the development testing.

Orig Findable: Original Findable SPRs or that subset of the original development SPRs that the project consultant deemed could be found during unit testing. (This category is an artifact due to original development testing not being conducted at the unit level, but at a higher "software build" level. Thus some errors found during original development testing and mapped to a sample may only be findable when testing interfaces.)

Exper Findable: Experiment findable SPRs, or the sum total of all known errors for a sample that could have been found by unit-level testing. (Equal to the sum of "Newly Found in Exper" + "Orig Findable.")

% of Exper Findable Found in Exp: The percent of the SPRs deemed findable that were actually found during our experimental testing by any one or more test techniques applied in the experiment.

(See also the Unit Error Summary Tables in Section 5.2.6; they present a more detailed view of these SPRs.)

As shown in Table 5.2, the units contained from one to twenty-one known errors that could possibly be found by unit testing; unit testing found from none to 100% of these errors, with an average percentage found of approximately 75% across the eight units. Thus while on average three quarters of the errors in a sample were found, the variance (from 0 to 100%) is large. Also, the number of errors known per sample

has a large variance (from 1 to 21). These large variances and somewhat small absolute numbers are less than ideally suited for statistical analyses. The analyses below thus assume the data are adequate; repetition of this study to add data points to the analyses would improve confidence in results.

It is also interesting to note that for the most part the experiment unit testing found new errors not previously found in original development testing. Conversely, experiment testing did not find many of the original development errors. This observation leads to two interesting hypotheses:

- a. The difference may point out that more formal unit testing should be advocated to find errors earlier in the development life cycle.
- b. The difference may point out that techniques used in this experiment were different and found different errors than techniques used in the actual development.

Item a) above is raised because the original development SPRs were logged against system "build" testing, not unit level testing. In iterative system builds, a harness surrounded the incomplete system, which included completed pieces of code with uncompleted sections stubbed out. Thus unit level testing was never performed on the units of the systems (unless done informally by programmers, and this was not documented). An interesting corollary to the issue making unit testing more formal is whether any of the new errors were critical errors. Investigating this question is suggested for future work.

Item b) is likely not the case. Documentation from original system testing implies that mainly functional testing was used. As noted in the above paragraph, it was performed on a higher level than the unit level. So we conclude the test level (and perhaps testers) was likely a more important factor than the techniques used in explaining the difference between experiment and original development errors found.

Whereas Table 5.2 addresses overall test effectiveness from a code sample viewpoint, Table 5.3 addresses effectiveness from the individual test technique viewpoint. This table shows that averaged over all samples at the unit test level, Code Review found the largest percentage of errors, followed by Error and Anomaly Detection, Branch, Functional, Random, and Structure Analysis. This descriptive data suggest that Code Review may be the best error-finding technique at the unit level. Since the experiment was designed as a latin square that allows us to test this hypothesis while accounting for other influences upon the data, such as tester variability, order of application of the techniques, it is premature to make this as a final judgment. See Section 5.2.2, which documents initial analysis of variance (ANOVA) results.

Table 5.3: Percent Findable SPRs Found by Test Technique

Level	B	R	F	C	EA	SA
Unit	25%	19%	23%	36%	29%	3%
CSC	68%	17%	33%	35%	25%	10%

Legend

B	Branch Testing	C	Code Review
R	Random Testing	EA	Error & Anomaly Detection
F	Functional Testing	SA	Structure Analysis

Single Technique Effort

Tables 5.4 and 5.5 show the wall clock time in hours needed, by each tester on each sample to set up for and execute test techniques during the experimental runs. In these tables, the field "Initial DTM Driver Development" represents the time needed for any one of the dynamic techniques only (B, F, R) to develop a test driver and to prepare the DEC Test Manager (DTM) tool for test execution. Thus, comparing time values among the techniques in this table shows time spent using the technique but not writing drivers, setting up the online environment, and so forth. This is one way of comparing time between the dynamic and static techniques listed in the table. The column labeled "Estimated Hours" shows the amount of time the principal investigators estimated each technique would take prior to the experiment runs.

These tables show that applying the techniques took approximately the same relative amount of time across samples, but that the SCENE sample did take more time in general than AFATDS. This may be accountable to the larger size and complexity of the SCENE unit samples.

In comparing the "Estimated Hours" with the observed "Average Effort" entries for all but one comparison, the observed was slightly less than estimated. The application of Code Review on the AFATDS project samples took on average over 20 hours -- well over the 8 hours estimated. A look at the individual applications shows that Tester II spent much more time on code review than the others. This was due to the tester having trouble interpreting when he had adequately addressed items on the checklist. No other tester had this difficulty; an apparent difference in approach to problem solving and/or judging "completion" of a code review task may account for this outlying data point. See also the section on Tester Profiles (Section 5.2.1.4).

Table 5.6 shows data from the two previous tables reduced into one table; data values represent the average time for a tester to test two samples with a given technique (or to set up two drivers and online test environments).

Tables 5.7 and 5.8 present test time in a different way than the two previous tables. In these tables, the field "Initial DTM Driver Development" has been added to the technique time value, for the dynamic techniques. Thus entries in this table reflect how much time it would have *actually* taken a tester to test with any one technique independently of any others in this experiment. Thus they can be recommended as a starting point for estimating time to test other, similar code units. Embedded in these times are the fact that the testers used the DTM tool and developed drivers for code of a given complexity; another test support tool and code of different complexities might necessitate altering these times.

Table 5.9 shows data from the two previous tables reduced into one table; data values represent the average time for a tester to test two samples with a given technique. Note that the average effort across eight samples for F, C, B, and R test techniques and across six samples for the E and SA test techniques is higher than the principal investigators originally estimated. The original estimates were good at predicting test time as shown above in Table 5.6; thus, the estimates seem to best reflect test application time excluding driver and online environment development. Additions for these activities can be separately estimated and added to the existing estimates, taking into account the driver complexity and number and types of tools used in setting up the test environment.

Comparing application effort across techniques shows that the static techniques took much less time than the dynamic techniques. Of the three static techniques, two

Tables 5.4 AFATDS UNIT DATA: SETUP & EXECUTION EFFORT

AFTDS PROJECT						
Unit		THUSCN	SNWMSN	THUADC	THNRDT	
Tester		I	II	III	IV	AVERAGE EFFORT
	Estimated Hours*					
Initial DTM Driver Dev.	N/A	35.25	32.50	19.00	9.50	24.06
F	16.00	24.00	3.75	3.50	5.00	9.06
C	8.00	7.25	64.00	6.00	3.50	20.19
B	29.00	15.00	2.00	7.00	12.00	9.00
R	22.00	10.75	18.50	8.50	8.00	11.44
E	6.0	N/A	N/A	0.25	1.00	.63
SA	4.0	N/A	N/A	0.50	0.25	.38

Tables 5.5 SCENE UNIT DATA: SETUP & EXECUTION EFFORT

SCENE PROJECT						
Unit		SCANNER	INFANSEN	INPSAT	INMODE	
Tester		I	II	III	IV	AVERAGE EFFORT
	Estimated Hours*					
Initial DTM Driver Dev.	N/A	35.50	13.75	16.25	7.00	18.13
F	16.00	23.50	6.75	14.20	18.00	15.61
C	8.00	8.00	5.75	8.00	4.00	25.75
B	29.00	35.75	31.50	21.50	11.50	25.06
R	22.00	16.00	16.25	17.25	16.50	16.50
E	6.0	.50	N/A	6.00	N/A	3.25
SA	4.0	.50	N/A	4.00	N/A	2.25

Tables 5.6 ALL UNIT DATA: AVERAGE SETUP & EXECUTION EFFORT

Unit		THUSCN	SNWMSN	THUADC	THNRDT	
Tester		SCANNER	INFANSEN	INPSAT	INMODE	
	Estimated Hours*	I	II	III	IV	AVERAGE EFFORT
Initial DTM Driver Dev.	N/A	35.36	23.13	17.63	8.25	21.10
F	16.00	23.75	5.25	8.85	11.50	12.34
C	8.00	7.63	34.88	7.00	3.75	13.32
B	29.00	25.38	16.75	14.25	11.75	17.03
R	22.00	13.38	17.38	12.88	12.25	13.97
E	6.0	0.25	N/A	3.13	0.50	0.97
SA	4.0	0.25	N/A	2.25	0.13	0.66

Table 5.7. AFATDS UNIT DATA : APPLICATION EFFORT

AFATDS PROJECT						
Unit		THUSCN	SNWMSN	THUADC	THXRDT	
Tester		I	II	III	IV	AVERAGE
Estimated Hours*						EFFORT
F	16.00	59.25	36.25	22.50	14.50	33.13
C	8.00	7.25	61.00	6.00	3.50	20.19
B	29.00	50.25	34.50	26.00	21.50	33.06
R	22.00	46.00	51.00	27.50	17.50	35.50
E	6.0	N/A	N/A	0.25	1.00	.63
SA	4.0	N/A	N/A	0.50	0.25	.38

Table 5.8. SCENE UNIT DATA : APPLICATION EFFORT

SCENE PROJECT						
Unit		SCANNER	INFANSEN	INPSAT	INMODE	
Tester		I	II	III	IV	AVERAGE
Estimated Hours*						EFFORT
F	16.00	59.00	20.50	30.45	25.00	33.74
C	8.00	8.00	5.75	8.00	4.00	6.44
B	29.00	71.25	45.25	37.75	18.50	43.19
R	22.00	51.50	30.00	33.50	23.50	34.63
E	6.0	50	N/A	6.00	N/A	3.25
SA	4.0	50	N/A	4.00	N/A	2.25

Table 5.9. ALL UNIT DATA : AVERAGE APPLICATION EFFORT

Units		THUSCN	SNWMSN	THUADC	THXRDT	AVERAGE
Tester		SCANNER	INFANSEN	INPSAT	INMODE	
Technique	Estimated Hours*	I	II	III	IV	EFFORT
F	16.00	59.13	28.38	26.48	19.50	33.37
C	8.00	7.63	34.88	7.00	3.75	13.32
B	29.00	60.75	39.88	31.88	20.00	38.13
R	22.00	48.75	40.50	30.50	20.50	35.07
E	6.00	0.25	N/A	3.13	0.50	0.97
SA	4.00	0.25	N/A	2.25	0.13	0.66

(E and SA) were fully automated and thus took very little time; conversely, code review involved manually reviewing the code against a checklist. Branch testing seemed to take the most time on average, followed by Random and then Functional.

Statistical tests were conducted to provide a more rigorous interpretation than this reduced data allowed: an ANOVA for the application effort data in the latin square is contained in Section 5.2.2. Also, an ANOVA was conducted for technique efficiency, with efficiency defined as technique effectiveness relative to technique effort. This analysis is also contained in Section 5.2.2.

Single Technique Branch Coverage

For all testing at the unit level, the code under test was instrumented with RXVP80. RXVP80 recorded the branch coverage during test execution, to three decimal places. Table 5.10 shows the percent of branches, as identified by RXVP80 for a sample, that were executed by a given technique for each sample.

Table 5.10: UNIT DATA: BRANCH COVERAGE BY TECHNIQUE

Units	THUSCN SCANNER	SNWMSN INFANSEN	THUADC INPSAT	THXRDT INMODE	AVERAGE COVERAGE
Tester	I	II	III	IV	
Technique					
F	1.00 810	500 787	1.00 865	935 585	810
B	1.00 899	1.00 898	1.00 865	1.00 792	932
R	889 924	875 620	875 729	968 585	808

The four samples listed on the first row in the table are from the AFATDS project, while the four listed in the second row are from the SCENE project. Coverage values in the cells are listed in a respective fashion. It appears that higher coverage may have been achieved on the AFATDS samples than the SCENE samples, in general. Average branch coverage was highest (over 93%) for branch testing; functional testing achieved slightly higher coverage (81%) than did random testing (just under 80%), although whether this is really a difference is questionable. One would expect branch testing to achieve high coverage, since branch coverage is an explicit goal and forms the basis of the stopping rule for this technique (100% of every executable branch executed twice was the primary stopping rule in this study). (See Table 5.11, which documents the stopping rules for each technique.) It is interesting to note that functional and random did as well as they did, since neither has a stopping rule explicitly based on branch coverage. These results are shown in Figure 5.3a.

Single Technique Efficiency

Test technique efficiency is the percent of findable SPRs found when the stopping rule was reached, divided by the time taken applying the technique when the stopping rule was reached. *For the experiment*, at the unit level the techniques rank in decreasing efficiency as follows: Error & Anomaly Detection, Structure Analysis, Code Review, Functional Testing, Branch Testing and Random Testing. A plot of unit

Table 5.11: Stopping Rules

TEST TECHNIQUE	STOPPING RULE	TEST LEVEL	
		Unit	CSC*
Branch Testing	100 % of branches executed (with a minimum of 2 traversals per branch) and MTTF = 10 input cases. Not to exceed X hours.	X = 29	X = 58
Code Review	All required aspects of the method have been evaluated using SDDL where possible, manually where not. Not to exceed X hours.	X = 8	X = 16
Functional Testing	All test procedures executed. Not to exceed X hours.	X = 16	X = 32
Random Testing	Minimum number, Y, samples from input space executed, and MTTF = 10 input cases. Not to exceed X hours.	X = 22 Y = 25	X = 44 Y = 50
Error & Anomaly Detection	All required aspects of the method have been evaluated, using automated tool where possible, manually where not. Not to exceed X hours.	X = 6	X = 12
Structure Analysis	All required aspects of the method have been evaluated, using automated tool where possible, manually where not. Not to exceed X hours.	X = 4	X = 8

Note: Unit test level stopping rules were used for CSC testing, due to budget constraints.

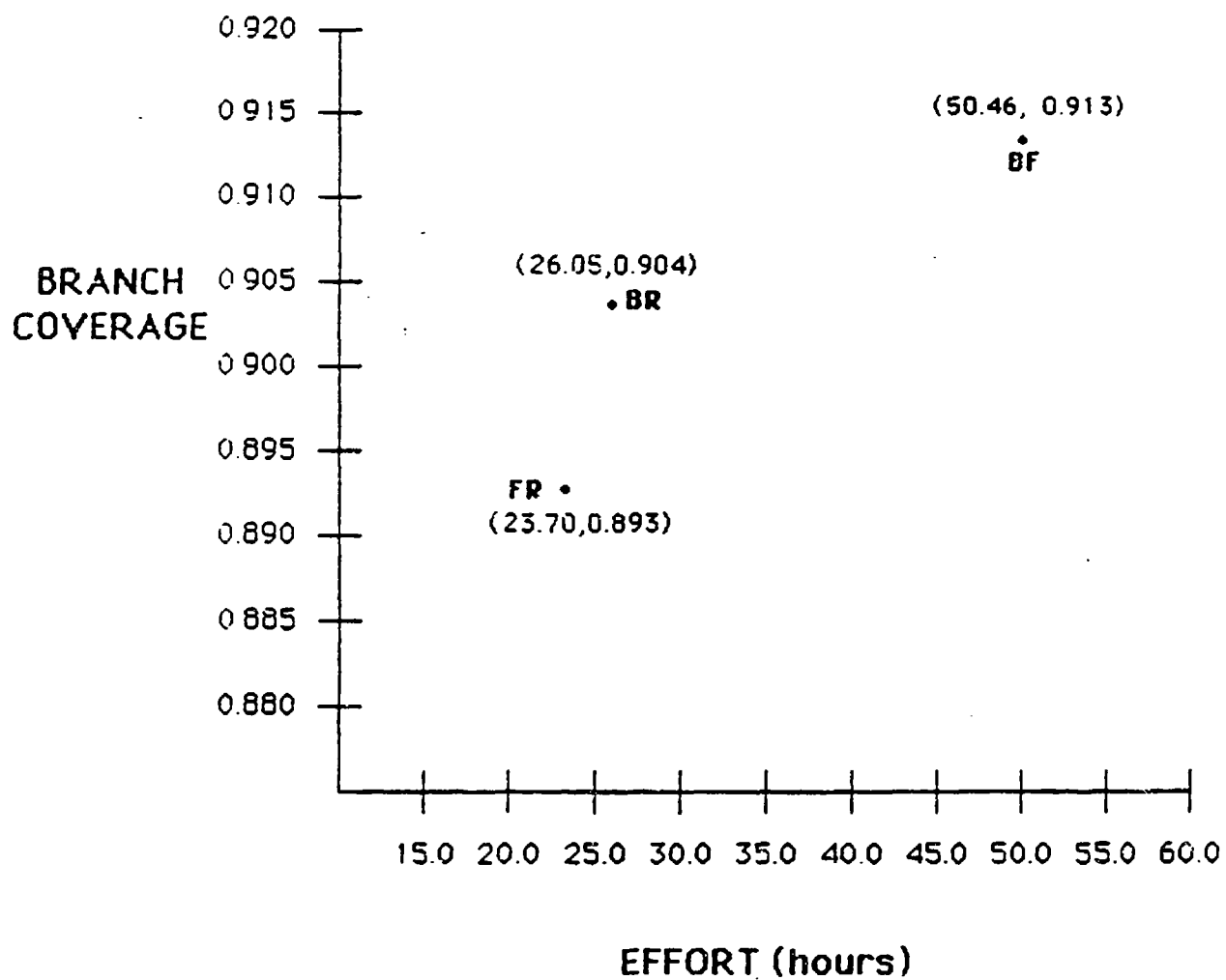


Figure 5.3a Paired Unit Data: Effort versus Coverage

level efficiency is given in Figure 5.3b. The only trend or relationship apparent here is that static techniques found a larger percentage of the known errors per unit time than did the dynamic techniques.

5.2.1.2.2 Test Technique Pair Description

Paired Technique Effectiveness

As shown in Table 5.12, the six test techniques: Branch (B), Code Review (C), Functional (F), Random (R), Error and Anomaly Analysis (E), and Structure Analysis (S) can be combined to represent 15 technique pairs. Technique error-finding effectiveness, as denoted by percent of SPRs found, is shown for each technique pair applied to each sample. The rightmost column shows the average percent of SPRs found for each technique pair, across all samples.

In decreasing effectiveness, the pairs are: CE, FE, BE, CR, RE, CF, CS, BC, ES & FR, BR, FS, BF, BS, and RS. Thus it appears that while Code Review did the best at finding errors at the unit level, it formed the most effective pair combined with Error & Anomaly Detection. The static technique of Error & Anomaly Detection combined with Functional or Branch testing did almost as well as the CE pair. Structure Analysis did not do well by itself and did not rate high in the pair analysis; it is interesting to note, however, that it performed best when paired with another static technique, Code Review. Of the pairs of dynamic techniques, FR found the most with 39%, followed by BR with an average of 38%, and BF with an average of 36%. Note that these dynamic pairs are rated in inverse order from the CSC level pairs. (See Table 5.19.)

Paired Technique Effort

Based on the single technique effort results, which showed static techniques took less time than the dynamic ones, technique pairs of two static techniques will also take less time than other pairs of techniques. This finding is reflected in Table 5.13. The pairs, listed in order of increasing effort necessary to apply a technique pair, are: ES, CS, CE, RS, RE, CF, FR, FS, CR, FE, BS, BE, BR, BF, and BC.

An ANOVA on pairs of techniques is not wholly appropriate from a statistical standpoint. Conducting a chi-square test on the data is one option for further analyses. A first iteration on conducting a chi-square test on these paired technique effort data is contained in the Task 5 report.

Paired Technique Coverage

Table 5.14 shows the average branch coverage attained by each pair of dynamic techniques. (Static techniques cannot be included here because their application does not involve executing the code, and therefore executing branches.) In direct translation from the single technique coverage results, BF obtained the highest coverage with over 91%, followed by BR with approximately 90% and FR with approximately 89%.

An ANOVA on pairs of techniques is not wholly appropriate from a statistical standpoint. Conducting a chi-square test on the data is one option for further analyses. A first iteration on conducting a chi-square test on these paired technique effort data is contained in the Task 5 report.

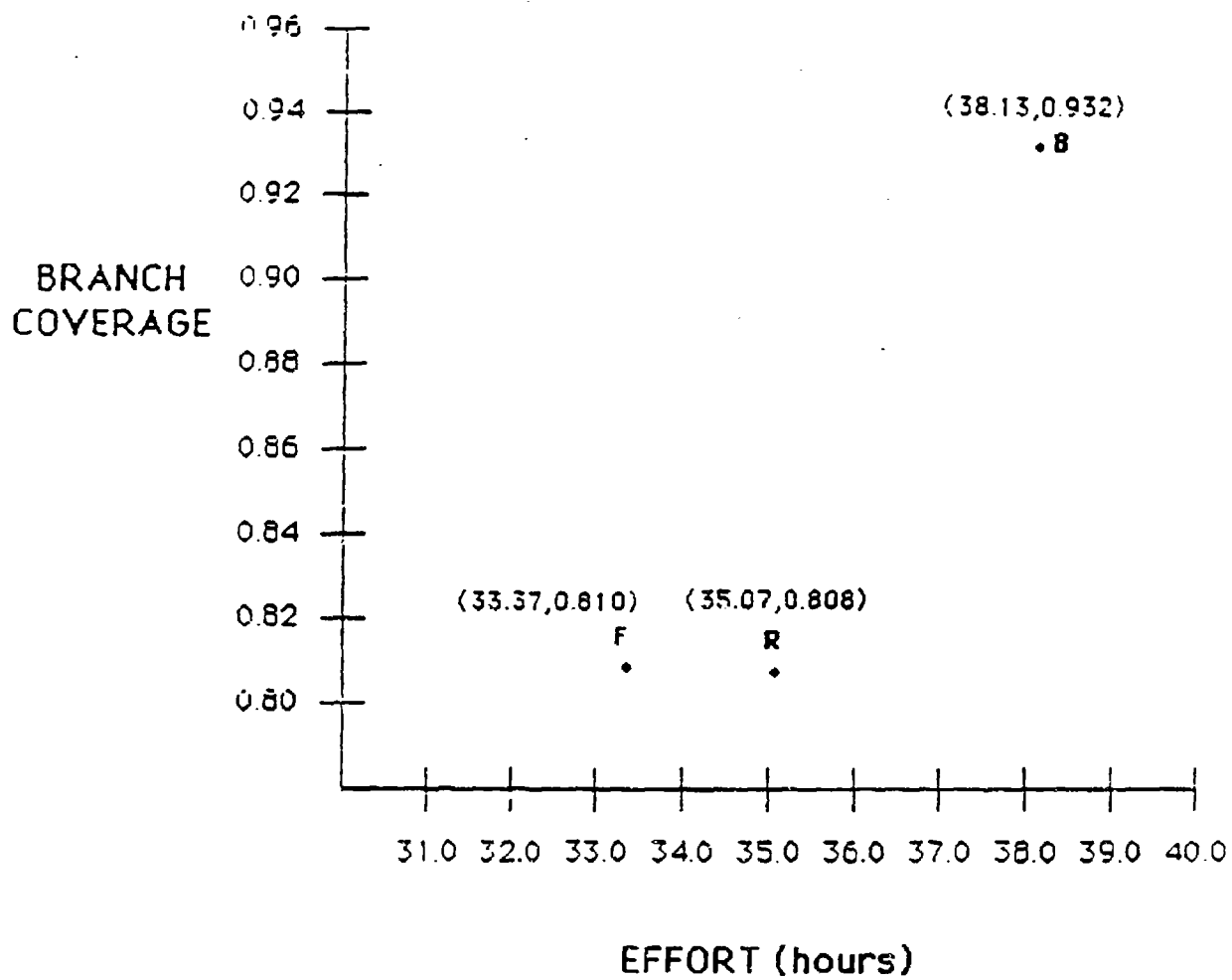


Figure 5.3b Single Technique Data: Unit Test Level Coverage vs Time

Table 5.12: Unit Paired Technique Effectiveness (7 SPRs)

Technique Pair	THUSCN SCANNER	SNWMSN INFANSEN	THUADC INPSAT	THXRDT INMODE	AVERAGE PERCENT
Tester	I	II	III	IV	
BC	0.0	0.50	0.67	0.79	0.43
	0.11	0.60	0.71	0.67	
BF	0.0	0.25	0.33	0.57	0.36
	0.0	0.50	0.57	0.62	
BR	0.0	0.50	0.33	0.50	0.38
	0.06	0.60	0.50	0.52	
BE	N/A	N/A	0.33	0.50	0.59
	0.72	N/A	0.79	N/A	
BS	N/A	N/A	0.33	0.43	0.33
	0.06	N/A	0.50	N/A	
CF	0.0	0.50	1.0	0.79	0.52
	0.11	0.20	0.71	0.81	
CR	0.0	1.0	1.0	0.79	0.58
	0.17	0.40	0.57	0.67	
CE	N/A	N/A	0.67	0.43	0.64
	0.83	N/A	0.64	N/A	
CS	N/A	N/A	0.67	0.43	0.44
	0.17	N/A	0.50	N/A	
FR	0.0	0.75	0.33	0.57	0.39
	0.06	0.20	0.50	0.67	
FE	N/A	N/A	0.33	0.57	0.62
	0.72	N/A	0.86	N/A	
FS	N/A	N/A	0.33	0.50	0.37
	0.06	N/A	0.57	N/A	
RE	N/A	N/A	0.33	0.50	0.53
	0.78	N/A	0.50	N/A	
RS	N/A	N/A	0.33	0.43	0.27
	0.11	N/A	0.21	N/A	
ES	N/A	N/A	0.33	0.07	0.39
	0.78	N/A	0.36	N/A	

Table 5.13 Unit Paired Technique Effort

Technique Pair	THUSCN SCANNER	SNWMSN INFANSEN	THUADC INPSAT	THXRDT INMODE	AVERAGE EFFORT
Tester	I	II	III	IV	
BC	136.75	149.50	77.75	47.50	51.44
BF	169.00	90.25	81.45	63.00	50.46
BR	148.25	114.50	89.50	64.50	26.05
BE	122.00	N/A	70.00	41.00	25.89
BS	122.00	N/A	68.25	40.25	25.61
CF	133.50	126.50	66.95	47.00	23.37
CR	112.75	150.75	75.00	48.50	24.19
CE	15.75	N/A	20.25	8.50	4.95
CS	15.75	N/A	18.50	7.75	4.67
FR	145.00	91.50	78.70	64.00	23.70
FE	118.75	N/A	59.20	40.50	24.27
FS	118.75	N/A	57.45	39.75	23.99
RE	98.00	N/A	67.25	42.00	23.03
RS	98.00	N/A	65.50	41.25	22.75
ES	N/A	N/A	0.75	1.25	
	1.00	N/A	10.00	N/A	2.17

Table 5.14: PAIRED UNIT DATA : BRANCH COVERAGE BY TECHNIQUE

Units	THUSCN SCANNER	SNWMSN INFANSEN	THUADC INPSAT	THXRDT INMODE	AVERAGE COVERAGE
Tester	I	II	III	IV	
Technique					
BF	1.00 .962	1.00 .898	1.00 .865	.971 .708	.913
BR	1.00 .724	1.00 .898	1.00 .865	.871 .877	.904
FR	1.00 .975	.875 .806	1.00 .865	.871 .755	.893

In comparing the descriptive paired technique data for effort and coverage, it is seen that the coverage is inversely related to the effort: the technique-pair that took the longest to apply attained the best branch coverage, and vice versa. These results are shown in Figure 5.3c.

5.2.1.3 CSC Testing

As described in Section 5.2.1.1, one latin square was used in the experiment at the CSC level. The samples for use in this latin square were four CSCs from the AFATDS project: SNWMSN, THPCON, THUADS, and THURDT. See Section 5.2.1.1 for information on these samples. In addition to the latin square experiment, the two deterministic test techniques, Error & Anomaly Detection (E) and Structure Analysis (SA), were applied to two of the samples by two of the testers.

5.2.1.3.1 Single Test Technique Description

Single Technique Effectiveness

Table 5.15 shows the categories that SPRs fall into, and how many SPRs fell into each category, for each CSC sample. The numbers in the cells of the table represent SPR counts. Each column in this table provides information for the sample named at the heading of the column. Refer to the text accompanying Table 5.2 for a definition of the rows in this table.

(See also the Unit Error Summary Tables in Section 5.2.6; they present a more detailed view of these SPRs.)

As shown in Table 5.15, the CSCs contained from 2 to 17 known errors that could possibly be found by CSC testing, and CSC testing found from 20% to 100% of these errors with an average percentage found of approximately 75% across the four CSCs. Thus while on average three quarters of the errors in a sample were found, the variance (from 20 to 100%) is large. Also, the number of errors known per sample has a large variance (from 2 to 17). This mirrors the properties of the unit data, with the additional fact that the CSC level has only half the data points as the unit level. These factors make the data less than ideally suited for statistical analyses and unsuitable for an ANOVA. Repetition of this study to add data points to the analyses would improve confidence in results.

As at the unit testing level, CSC testing found new errors not previously found in original development testing. Conversely, CSC experiment testing found only one of the original development errors.

Whereas Table 5.15, addresses overall test effectiveness from a code sample viewpoint, Table 5.3 addresses effectiveness from the individual test technique viewpoint. This table shows that averaged over all samples, at the CSC test level, Branch Testing found the largest percentage of errors, followed by Code Review, Functional, Error and Anomaly Detection, Random, and Structure Analysis.

These descriptive data suggest that Branch Testing may be the best error-finding technique at the CSC level. Since the experiment was designed as a latin square that allows us to test this hypothesis while accounting for other influences upon the data, such as tester variability, and order of application of the techniques, it is premature to make this as a final judgment. Due to time and budget constraints, the minimum number (two) of latin squares needed to support an ANOVA at the CSC level was not obtained. Therefore we cannot conduct ANOVAs on these data. However, conducting

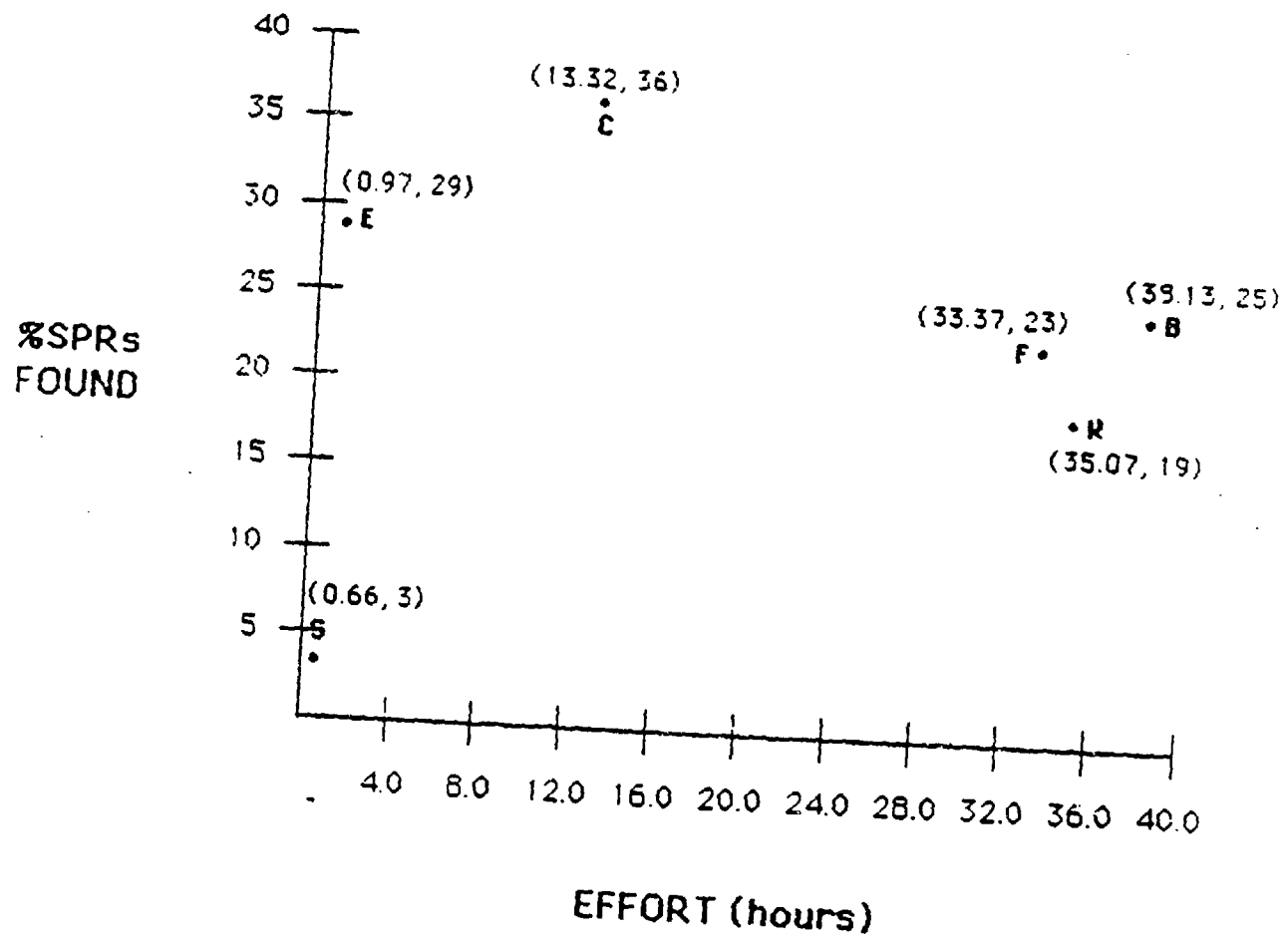


Figure 5.3c Single Technique Data: Unit Test Level Efficiency

Table 5.15: Breakdown of SPRs by CSC Sample

CSCs				
SPRs	SNWMSN	THPCON	THUADS	THURDT
Found in Experiment	6	16	2	2
Newly Found	6	16	1	2
Original Findable	1	1	4	0
Experiment Findable	7	17	5	2
% Experiment Findable, Found in Experiment	86%	94%	20%	100%

a chi-square test on the data is one option for further analyses. A first iteration on conducting a chi-square test on these CSC effort data is contained in the Task 5 report; further work needs to be done with these data analyses.

Single Technique Effort

Table 5.16 shows the average effort (in hours) to execute each technique on each sample. The times listed for dynamic techniques does not include driver development and online environment setup time in this table; this time is listed separately in the "Initial DTM Driver Development" fields.

Table 5.17 presents test time in a different way than the previous table. In this table, the field "Initial DTM Driver Development" has been added to the technique time value for the dynamic techniques. Thus entries in this table reflect how much time it would have *actually* taken a tester to test with any one technique independently of any others in this experiment. Thus they can be recommended as a starting point for estimating time to test other, similar code CSCs. The three dynamic techniques all took between 35 and 40 hours. Branch and Random tied for consuming the most effort, with Functional the third most time consuming. The static techniques all took much less time (2-7 hours) than the dynamic ones. Code Review took the most, with Error & Anomaly Detection and Static Analysis tied for the least amount of time. (Since the latter two techniques involve using error reports from the same automated tool, the equal average effort is not surprising.)

Testers did not use the DTM tool at the CSC test level, so DTM tool usage time is not embedded in these times as it was for the unit effort times. However, as with the unit effort, CSC times included time to develop drivers for code of a given complexity; code of different complexities might necessitate altering these times.

For both methods of measuring effort, the average effort, across four samples for F, C, B, and R test techniques and across two samples for the E and SA test techniques is lower than the principal investigators originally estimated. In opposition to what was seen at the unit level, the original estimates were better at predicting test time as shown above in Table 5.17; thus the estimates seem to best reflect test application time *including* driver and online environment development. This is likely because only partial CSCs were actually tested; the original estimates were made for CSC test effort in general.

Single Technique Coverage

Table 5.18 shows the branch coverage achieved by the three dynamic techniques at the CSC level. The relative ranking in decreasing coverage: B, F, R, mirrors the coverage performance seen at the unit level. (Refer to Table 5.10.) These results are shown in Figure 5.3d.

Single Technique Efficiency

Test technique efficiency is the percent of findable SPRs found when the stopping rule was reached, divided by the time taken applying the technique when the stopping rule was reached. *For the experiment*, at the CSC level the techniques rank in decreasing efficiency as follows: E, S, C, B, F, R. A plot of unit level efficiency is given in Figure 5.3e. The only trend or relationship apparent here is that static techniques found a larger percentage of the known errors per unit time than did the dynamic techniques.

Table 5.16: ALL CSC DATA : SETUP AND EXECUTION EFFORT

CSCs		THURDT	THUADS	SNWMSN	THPCON	AVERAGE
Tester		I	II	III	IV	EFFORT
Technique	Estimated Hours*					
Initial DTM Driver Dev.	N/A	55.50	11.50	24.00	9.50	21.13
F	32.00	11.00	7.00	10.75	6.50	8.81
C	16.00	8.50	5.00	8.00	3.50	6.25
B	58.00	19.50	9.50	16.25	7.00	13.06
R	44.00	16.00	13.25	12.75	10.25	13.06
E	12.00	N/A	N/A	4.00	1.50	2.75
SA	8.00	N/A	N/A	4.00	1.50	2.75

Table 5.17: ALL CSC DATA : TOTAL APPLICATION EFFORT

Units		THURDT	THUADS	SNWMSN	THPCON	AVERAGE
Tester		I	II	III	IV	EFFORT
Technique	Estimated Hours*					
F	32.00	66.56	18.50	34.75	16.00	33.94
C	16.00	8.50	5.00	8.00	3.50	6.25
B	58.00	75.00	21.00	40.25	16.50	38.19
R	44.00	71.50	24.75	36.75	19.75	38.19
E	12.00	N/A	N/A	4.00	1.50	2.75
A	8.00	N/A	N/A	4.00	1.50	2.75

Table 5.18: CSC DATA : BRANCH COVERAGE BY TECHNIQUE

CSCs	THURDT	THUADS	SNWMSN	THPCON	AVERAGE
Tester	I	II	III	IV	COVERAGE
Technique					
F	.561	.708	.773	.694	.684
B	.561	.792	.864	.837	.764
R	.366	.750	.864	.510	.623

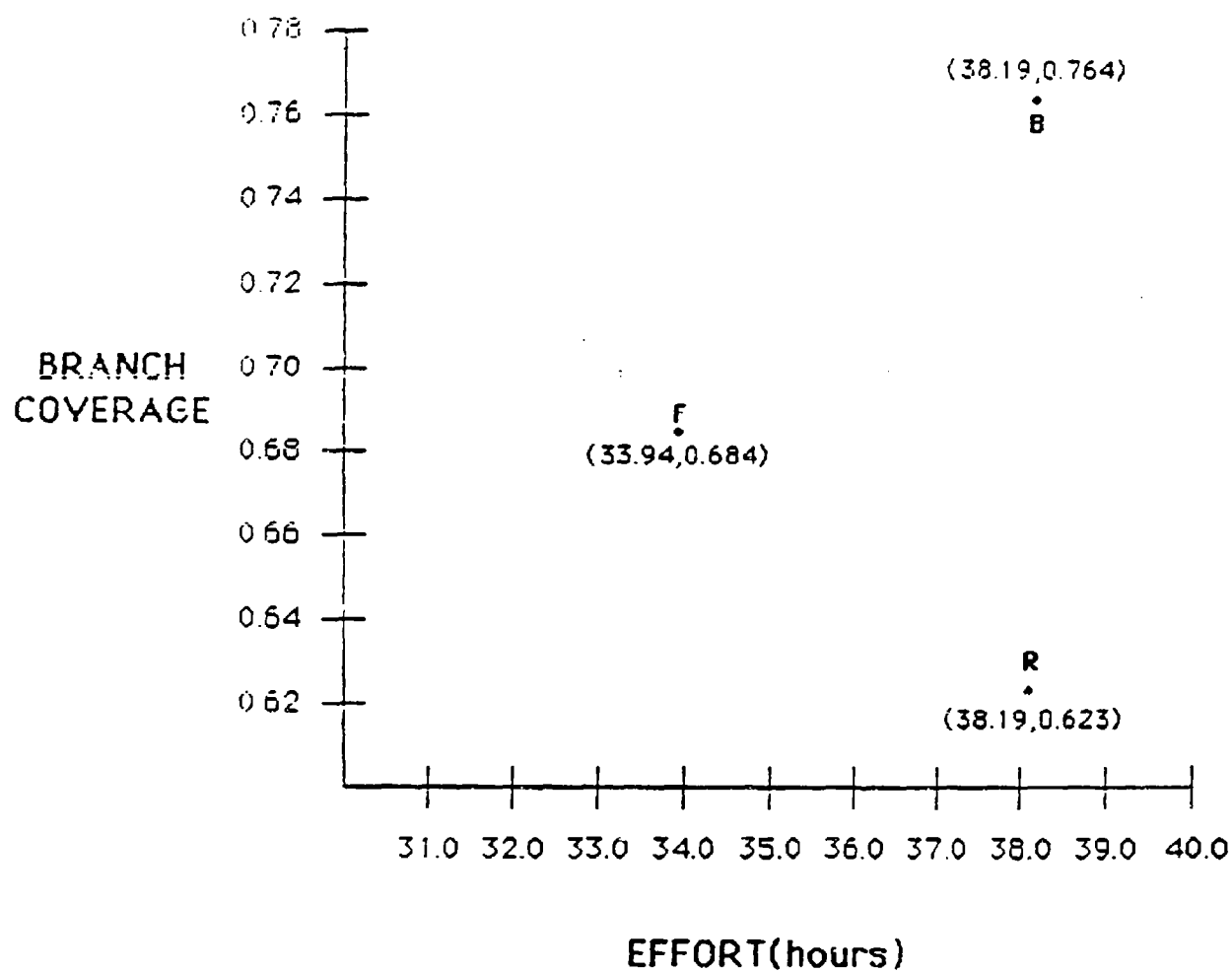


Figure 5.3d Single Technique Data: CSC Test Level Coverage vs Time

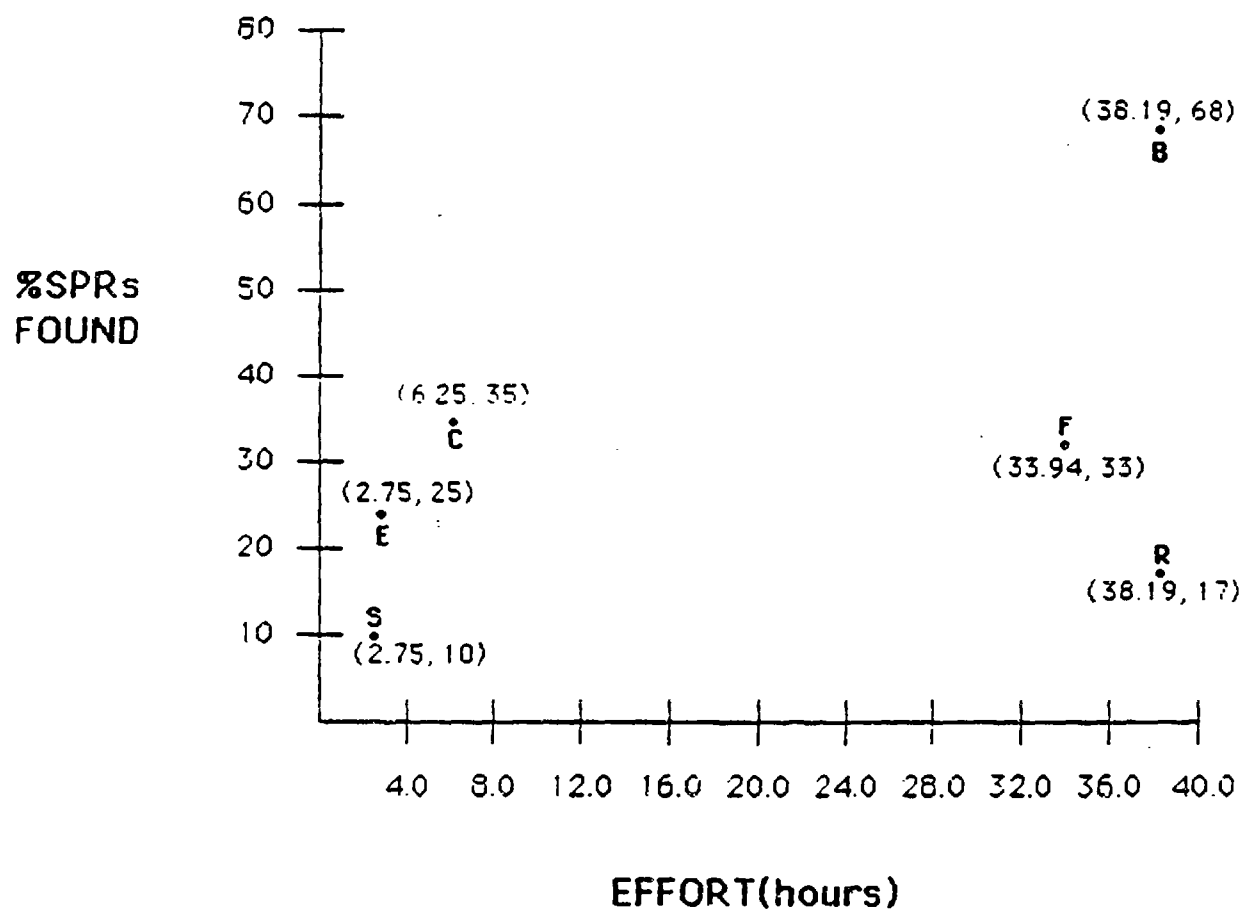


Figure 5.3e Single Technique Data: CSC Test Level Efficiency

5.2.1.3.2 Test Technique Pair Description

Paired Technique Effectiveness

As shown in Table 5.19, the six test techniques: Branch (B), Code Review (C), Functional (F), Random (R), Error and Anomaly Analysis (E), and Structure Analysis (S) can be combined to represent 15 technique pairs. Technique error-finding effectiveness, as denoted by percent of SPRs found, is shown for each technique pair applied to each sample. The rightmost column shows the average percent of SPRs found for each technique pair, across all samples.

In decreasing effectiveness, the pairs are: BE & BS, BC & BF, BR, FE, FS, CE & CS, CF, CR & RE & RS, FR, and ES. Thus it appears that while Branch Testing did the best at finding errors at the CSC level, it also complements errors found by the static techniques: Error & Anomaly Detection, Structure Analysis, and Code Review, in decreasing order. However, Error & Anomaly Detection and Structure Analysis did not do well by themselves at finding errors and did not complement each other well, which led to that pair (ES) rating the lowest. Of the pairs of dynamic techniques, BF found the most with 74%, followed by BR with an average of 69%, and FR with an average of 35%. Note that these pairs are rated in inverse order from the unit-level pairs. (FR found an average of 39%, followed by BR with an average of 38%, and BF with an average of 36% at the unit level.)

Paired Technique Effort

The single unit and CSC technique effort results showed static techniques took less time than the dynamic ones, and technique pairs of two static techniques also took less time than other pairs of techniques. This finding is reflected in Table 5.20. The pairs, listed in order of increasing effort necessary to apply a technique pair, are: ES, CE & CS, CF, CR, FE & FS, RE & RS, BE & BS, BC, BF & FR, and BR.

An ANOVA on pairs of techniques is not wholly appropriate from a statistical standpoint. Conducting a chi-square test on the data is one option for further analyses. A first iteration on conducting a chi-square test on these paired technique effort data is contained in the Task 5 report.

Paired Technique Coverage

Table 5.21 shows the average branch coverage attained by each pair of dynamic techniques. (Static techniques cannot be included here because their application does not involve executing the code, and therefore executing branches.) BF obtained the highest coverage with over 76%, followed by BR with approximately 72% and FR with approximately 73%.

An ANOVA on pairs of techniques is not wholly appropriate from a statistical standpoint. Conducting a chi-square test on the data is one option for further analyses. A first iteration on conducting a chi-square test on these paired technique coverage data is contained in the Task 5 report.

In comparing the descriptive paired technique data for effort and coverage, it is seen that the coverage is inversely related to the effort: the technique-pair that took the longest to apply attained the best branch coverage, and vice versa. These results are shown in Figure 5.4.

Table 5.19: CSC DATA: PAIRED TECHNIQUE EFFECTIVENESS (SPRs)

Technique Pair	THURDT	THUADS	SNWMSN	THPCON	AVERAGE
Tester	I	III	III	IV	PERCENT
BC	1.00	0.40	0.86	0.71	0.74
BF	1.00	0.20	0.86	0.88	0.74
BR	1.00	0.20	0.86	0.71	0.69
BE	N/A	N/A	0.86	0.76	0.81
BS	N/A	N/A	0.86	0.76	0.81
CF	0.00	0.40	0.86	0.65	0.48
CR	0.00	0.40	0.86	0.47	0.43
CE	N/A	N/A	0.71	0.35	0.53
CS	N/A	N/A	0.71	0.35	0.53
FR	0.00	0.20	0.57	0.65	0.36
FE	N/A	N/A	0.71	0.65	0.68
FS	N/A	N/A	0.57	0.65	0.61
RE	N/A	N/A	0.57	0.29	0.43
RS	N/A	N/A	0.57	0.29	0.43
ES	N/A	N/A	0.57	0.06	0.32

Table 5.20: CSC DATA: PAIRED TECHNIQUE EFFORT

Technique Pair	THURDT	THUADS	SNWMSN	THPCON	AVERAGE
Tester	I	III	III	IV	EFFORT
BC	83.50	26.00	48.25	20.00	44.44
BF	86.00	28.00	51.00	23.00	47.00
BR	91.00	34.25	53.00	26.75	51.25
BE	N/A	N/A	44.25	18.00	31.13
BS	N/A	N/A	44.25	18.00	31.13
CF	75.00	23.50	42.75	19.50	15.56
CR	80.00	29.75	44.75	23.25	17.00
CE	N/A	N/A	12.00	5.00	8.50
CS	N/A	N/A	12.00	5.00	8.50
FR	82.50	31.75	47.50	26.25	47.00
FE	N/A	N/A	38.75	17.50	28.13
FS	N/A	N/A	38.75	17.50	28.13
RE	N/A	N/A	40.75	21.25	31.00
RS	N/A	N/A	40.75	21.25	31.00
ES	N/A	N/A	8.00	3.00	5.50

Table 5.21: CSC DATA: PAIRED TECHNIQUE BRANCH COVERAGE

Technique Pair	THURDT	THUADS	SNWMSN	THPCON	AVERAGE PERCENT
Tester	I	III	III	IV	
BF	.561	.792	.909	.796	.765
BR	.476	.792	.864	.755	.722
FR	.561	.750	.909	.694	.729

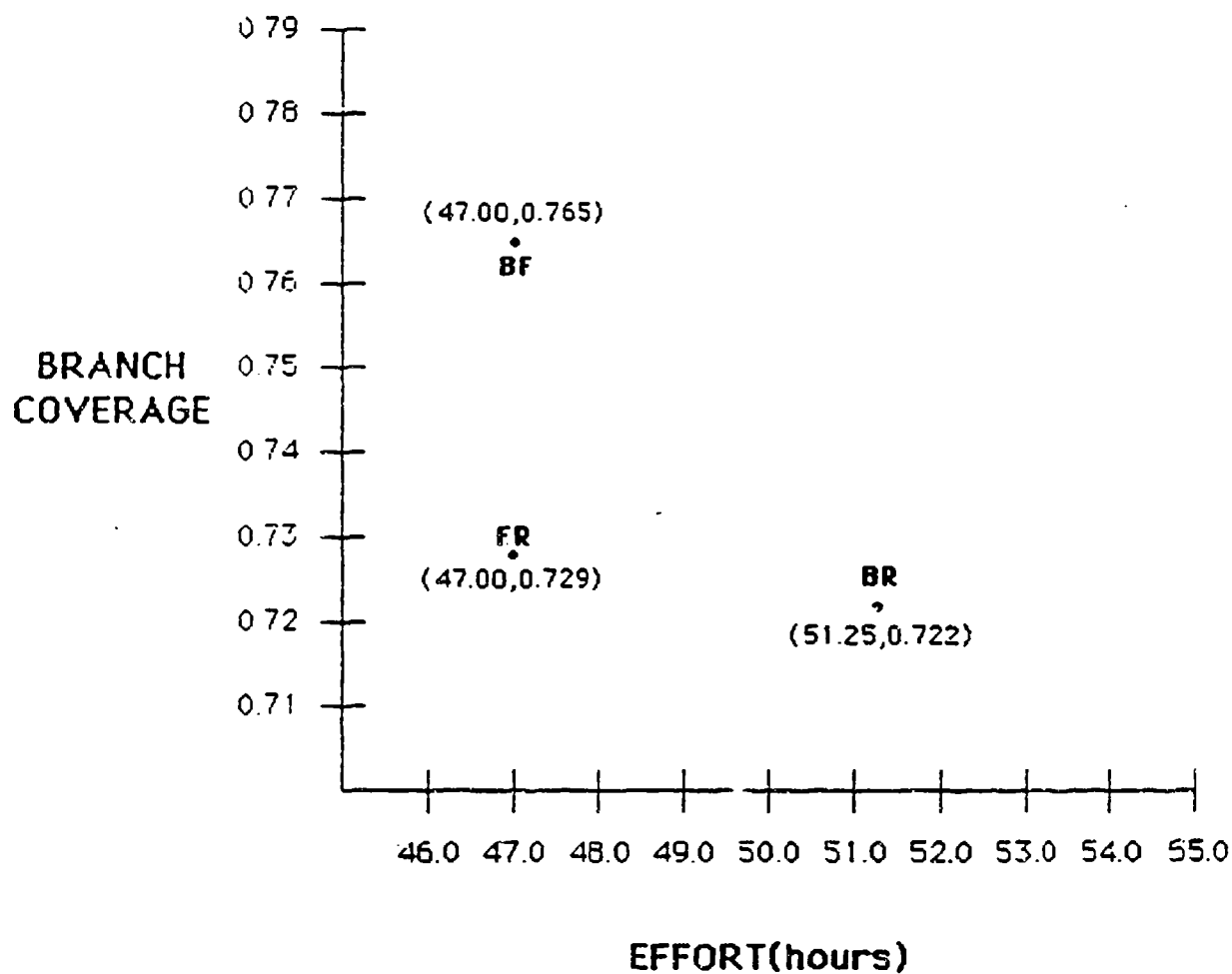


Figure 5.4: Paired CSC Data: Effort versus Coverage

5.2.1.4 Tester Profiles

Prior to testing software for this experiment, each tester was asked to complete a software engineering questionnaire. The results of this questionnaire are shown below in a series of histograms which compare tester backgrounds and experience.

In the education and experience category (see Figure 5.5), the four testers all have college degrees: one earned a Masters while the other three hold bachelors degrees. All have at least three years of general software experience and between one and five years of software testing experience.

Experience with test techniques used in this study is as follows (see also Figure 5.6 and 5.7): all use functional testing frequently; two had used random testing before and two had not; two use code review frequently, one uses it occasionally, and one had never used it; one occasionally uses structure analysis, while three had never used it; none had ever used error and anomaly detection, although one had read about it; and none had ever used branch testing, although two had read about it. (Interestingly enough, when interviewed after the experiment, all four testers liked branch testing. Their confidence in their results increased due to having the "concrete" measure of branch coverage against which to judge their progress, and they had no problems identifying test cases to exercise the branches.)

All had experience using high level languages (see Figure 5.8 and 5.9). One had used FORTRAN over 15 years, while one had used it under five years and two had never used it. Other languages one or more of the testers were familiar with include: C, Ada, Pascal, Assemblers, and others.

Testers for the most part did not have experience with the software tools used in this experiment (see Figure 5.10). None had used RXVP80, though all said they liked this tool after the experiment; only one had used DEC Test Manager (DTM). The testers were given time before the experiment began to familiarize themselves with both tools, in an effort to eliminate a tool learning effect in the results.

5.2.2 ANOVA Results

ANOVAs were conducted for the unit test level data, for hypotheses concerning single test techniques. The STATISTICAL ANALYSIS SYSTEM (SAS²) software tool's General Linear Models (GLM) procedure was used to automate the computations. ANOVAs run and interpreted on the unit level data include: test technique effectiveness (% of SPRs found), test technique branch coverage, test technique effort (hours), and test technique efficiency (SPRs found relative to the time it took to find them). The actual SAS output for these runs can be found in the Task 5 report.

Below is a description of the ANOVA model, followed by ANOVAs in each category listed above. Within each category is the analysis description taken from the Task 2 Report, (SRTIP) [1], Appendix B, and the interpretation of the SAS output results.

The analysis descriptions taken from the SRTIP have been modified to fit the terminology and expectations that have been adopted since its production. Refer to the SRTIP, Appendix B for an explanation of symbols and conventions used in these analysis descriptions. Refer to the document "Data Items Required for Analysis" in Appendix C of the Task 4 report [12] for definition of elements such as "total unique findable SPRs."

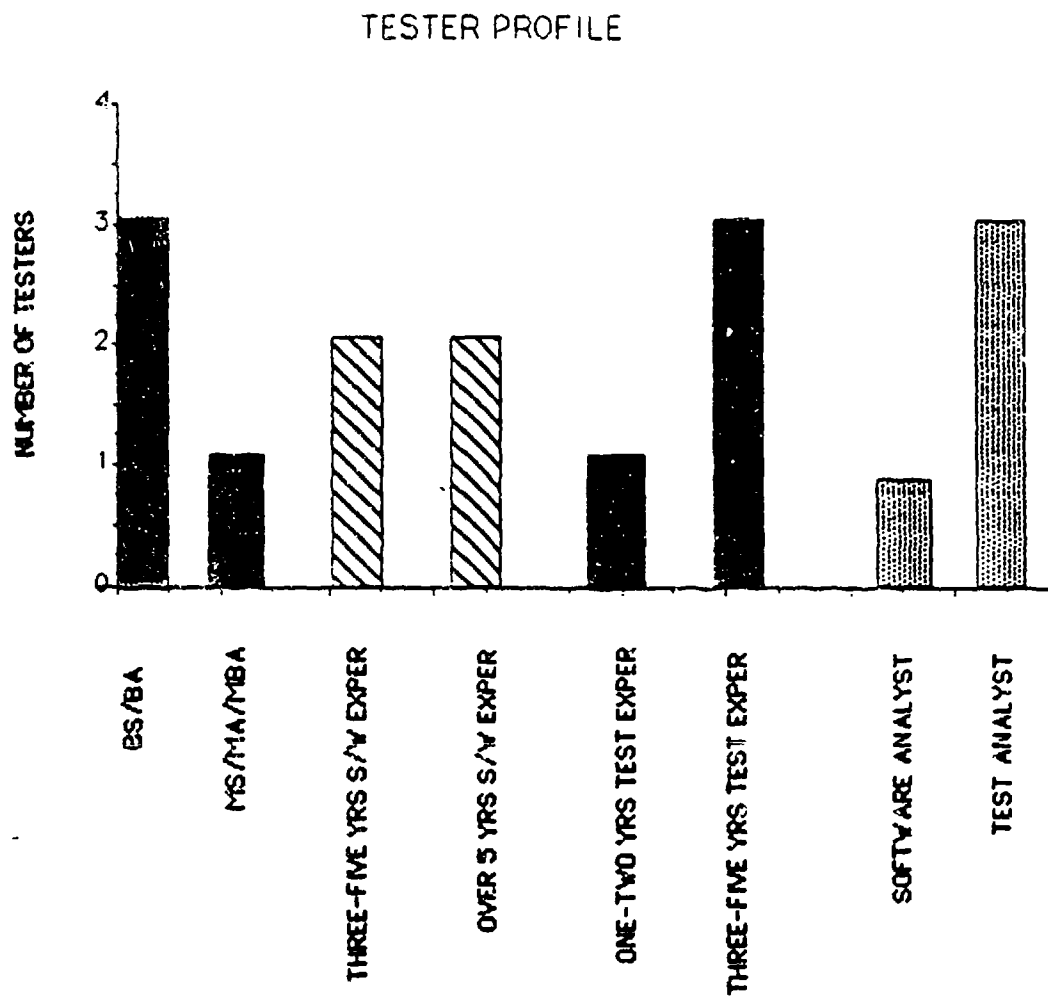


Figure 5.5: Tester Profile: Education and Work Experience

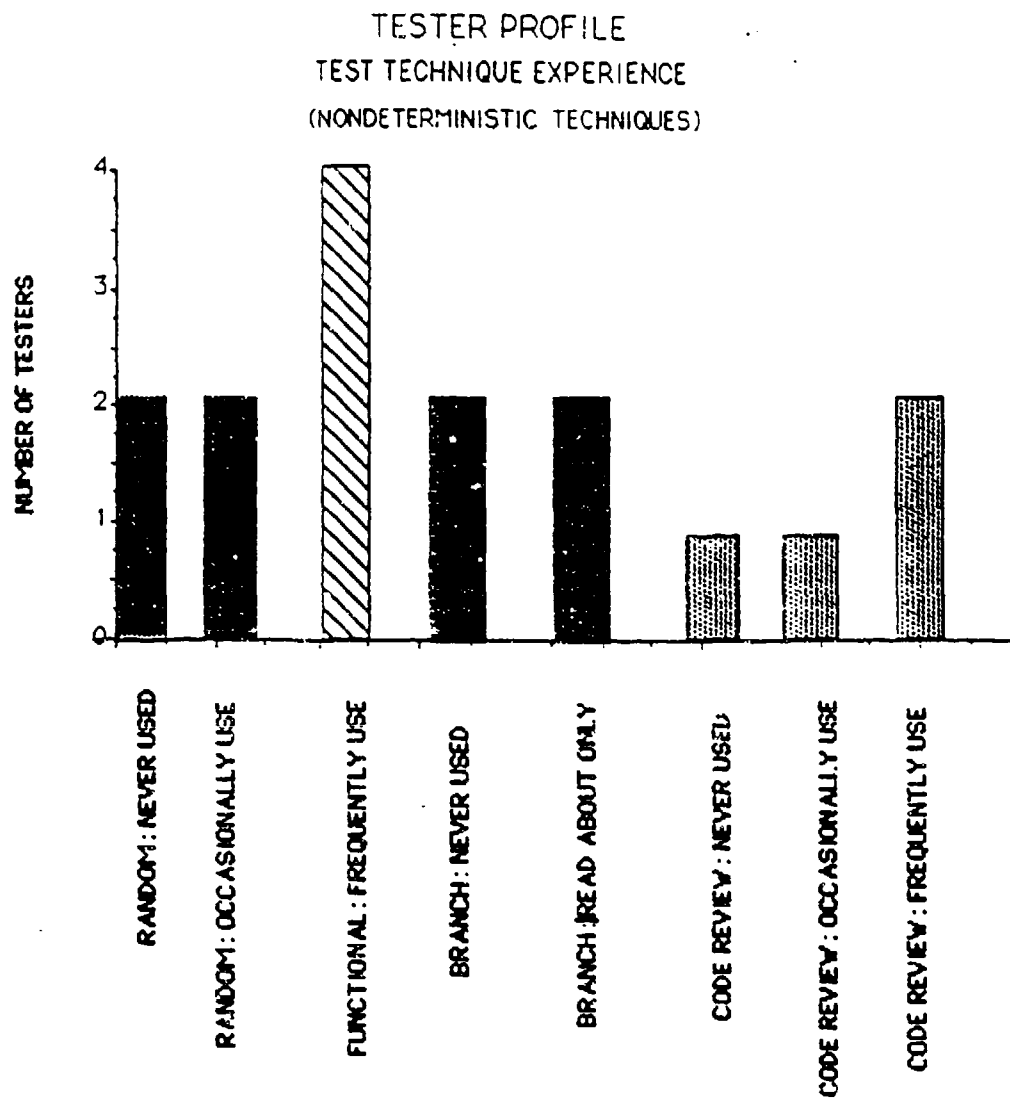


Figure 5.6: Tester Profile: Nondeterministic Test Techniques

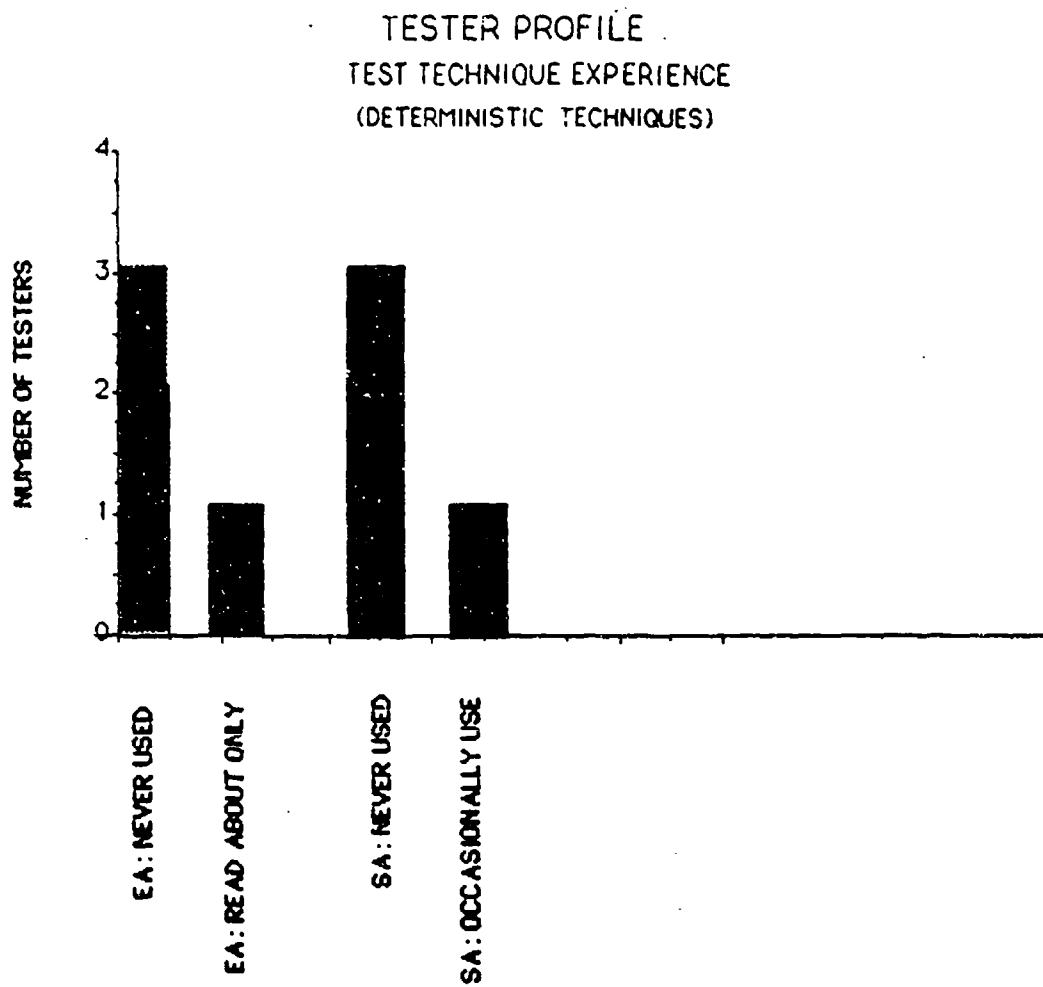


Figure 5.7: Tester Profile: Deterministic Test Techniques

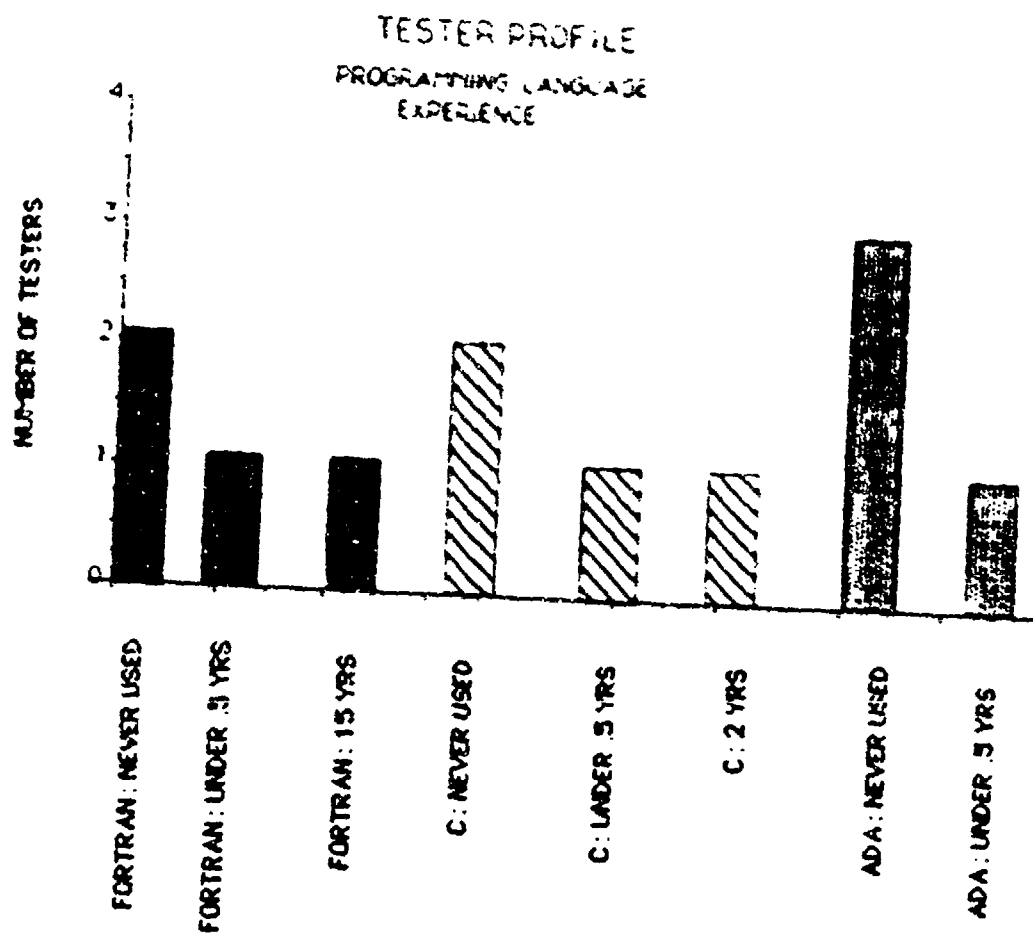


Figure 5.3: Tester Profile: Programming Language Experience

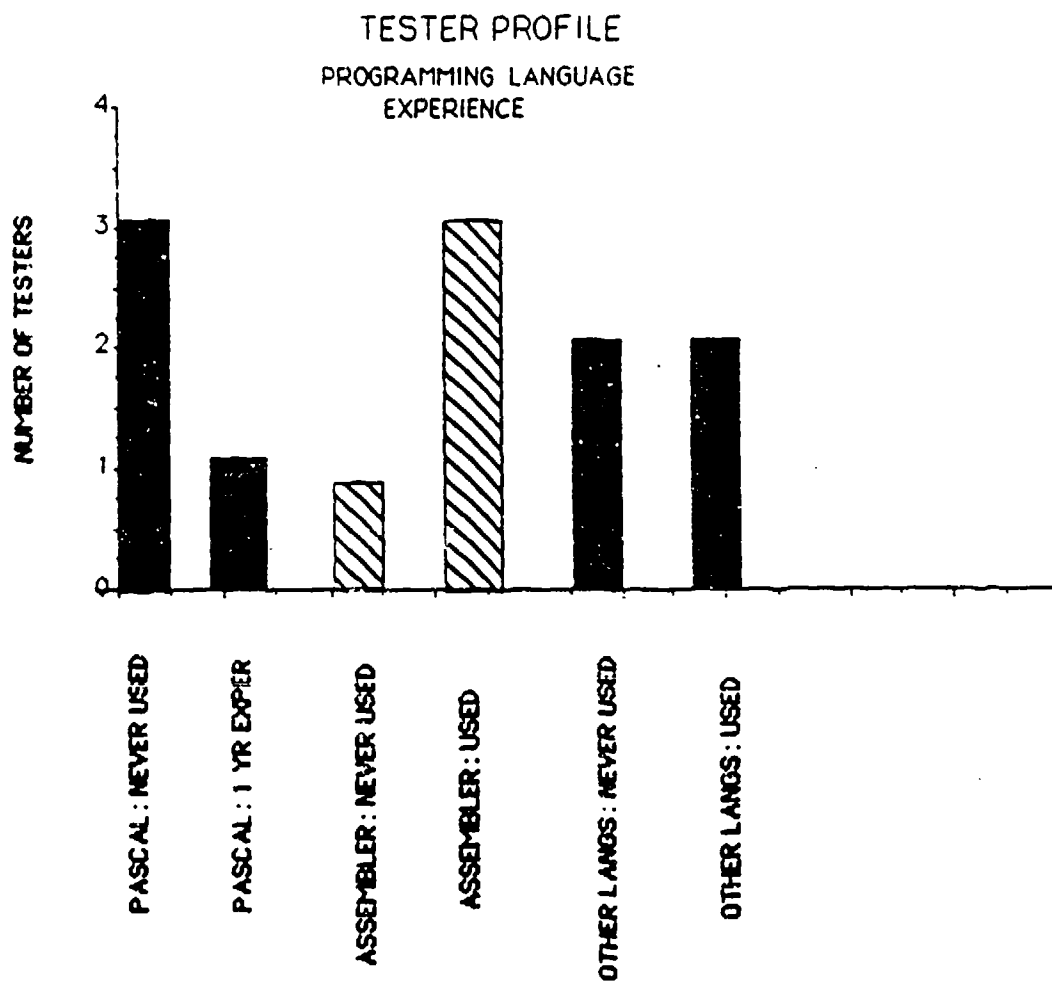


Figure 5.9: Tester Profile: Programming Language Experience (cont.)

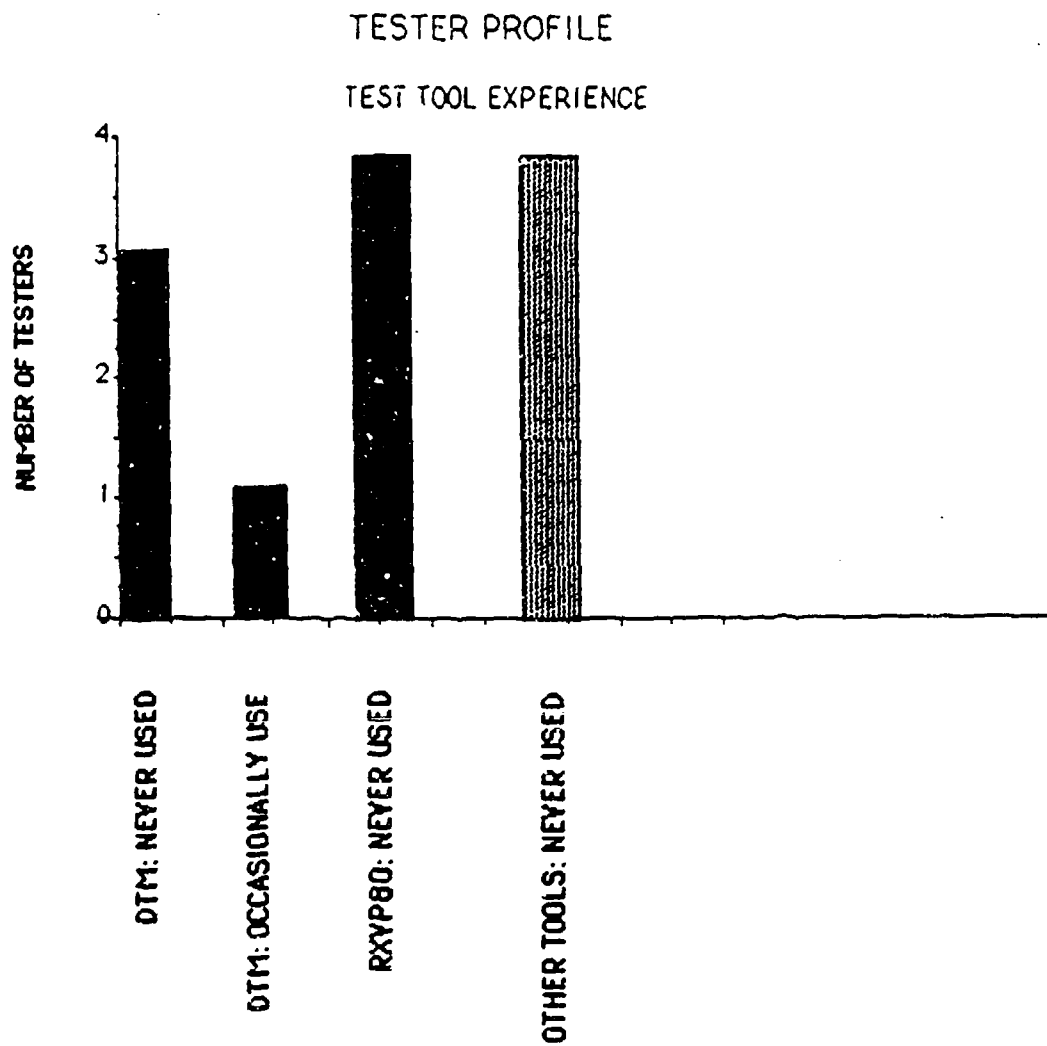


Figure 5.10: Tester Profile: Test Tool Experience

5.2.2.1 Process

To comparatively evaluate the effectiveness of the test techniques, the data collected during the experiment were analyzed using the ANOVA associated with this design, as described below:

$$Y_{ijk} = \mu = P_i + T_j + (PT)_{ij} + S_k + M_{ijk} + R_{ijk} + ijk$$

where

i is the project index

j is the tester/code sample index

k is the session index

P_i is the project effect

T_j is the combined tester and code sample effect

$(PT)_{ij}$ is the interaction effect among project, tester, and code sample

S_k is the test session effect

M_{ijk} is the test technique or method effect

R_{ijk} is the effect of the previously applied or residual test technique

ijk is the error term

The design used is that for estimating residual effects (in this case "learning" effects) when treatments are applied in sequence as described in Cochran and Cox [5] on pages 133-139.

In the ANOVA for the unit test level experiment, we have $m=2$ Latin Squares and $n=4$ treatments or test techniques with the sources of variation partitioned as shown in Table 5.22. Due to the nature of these experiments, we anticipate that if session, treatment, and residual effects are present, then they will be similar for each square; hence, it seems reasonable to pool the corresponding interactions into an error source of variation. On the other hand, because of the way the columns are defined, it would be appropriate to group the squares, columns, and squares by columns sources of variation into a single component (referred to as "columns" in the resultant ANOVA table given in Table 5.22) that can then be partitioned into the more meaningful subcomponents of testers and software project.

**Table 5.22 Unit Test Level Experiment:
Analysis of Variance**

Sources of Variation	Degrees of Freedom
Sequence or Columns	7
Tester Effect T_j	3
Software Project Effect	1
Testers x Software	3
Rows	
Test Session Effect S_k	3
Treatments	
Test Technique Effect M_{ijk}	3
Residual Effects	
Learning Effect R_{ijk}	3
Error Effect ϵ_{ijk} (assumes some errors are negligible)	15
TOTAL	31

5.2.2.2 Single Technique Effectiveness Results

Analysis Description from the SRTIP:

Identifier: TT_SINGLE_EFFECTIVENESS_01

Question of Interest: Is there a difference in the PERCENT TECHNIQUE SPRs* found by all testers using any one test technique (R, F, B, C, S, E) at the {unit, CSC} test level?

Analysis: Analysis of Variance Model F Test

NOTE: 1. * PERCENT TECHNIQUE SPRs = TECHNIQUE SPRs/

(ORIGINAL FINDABLE SPRs + TECHNIQUE NEW SPRs)

or 'the number of SPRs written for this technique' divided by 'the number of "findable" SPRs for the sample.'

2. The effect of the tools will be confounded.

Interpretation of SAS output:

DEPENDENT VARIABLE: P_SPR

FORMULA: Percent of known, findable at the current test level, SPRs that were detected by a technique. Specifically, SPRs detected by the technique divided by all unique experiment findable SPRs known (whether from original development or newly found during experiment).

F TEST FOR MODEL SIGNIFICANCE:

F	α	Significance?
2.90	.10	Yes
	.05	Yes
	.01	No

INTERPRETATION: Model is significant. Conduct all analyses at 95% confidence level or lower.

F TESTS FOR SOURCE SIGNIFICANCE (TYPE III):

SOURCE	F	Significance?		
		.10	.05	.01
SWTYPE	0.32	No	No	No
TESTER	10.11	Yes	Yes	Yes
SWTYPE*TESTER	2.22	No	No	No
SESSION	0.66	No	No	No
TECHNIQUE	1.79	No	No	No
RESIDUAL	2.58	Yes	No	No

INTERPRETATION: The tester effect is highly significant and the order of application is marginally significant and the order of application is marginally significant. This result indicates that once the tester variability is isolated, nothing can be said about technique effectiveness. Thus, the effectiveness at testing is based on the tester only and testers should be selected with this information in mind. This result also implies that we should continue to focus on defining test techniques and tools to minimize the tester's impact on reliability achievement.

T TESTS OF PAIRED MEANS: Avg. P_SPR's

INTERPRETATIONS:

- SWTYPE** • Not significantly different between projects.
- TESTER** • T1 has significantly fewer Avg. P_SPR's. This result may be due to code sample difficulty or tester skill level/experience.
• T2 performance is indistinguishable from T3 and T4.
• T4 performance is indistinguishable from T3.
- SESSION** • Not significantly different between sessions.
- TECHNIQUE** • Not significantly different between techniques.

5.2.2.3 Single Technique Coverage Results

Refer also to SAS data in the Task 5 report.

Analysis Description from the SRTIP:

Identifier: TT_SINGLE_COVERAGE_01

Question of Interest: Is there a difference in the percent of execution branches tested by each test technique (R, F, B) at the {unit} test level?

Analysis: Analysis of Variance Model F Test

Interpretation of SAS output:

DEPENDENT VARIABLE: PT_PATH

FORMULA: Percent of branches through the sample that a technique executed. Specifically, the number of branches executed by the test cases generated using a technique divided by number of branches in the code sample being tested.

F TEST FOR MODEL SIGNIFICANCE:

F	α	Significance?
30.77	.10	Yes
	.05	Yes
	.01	Yes

INTERPRETATION: Branch coverage for code reading assumed zero to avoid handling of missing cells. Model is highly significant.

F TEST FOR SOURCE SIGNIFICANCE:

SOURCE	F	Significance?		
		.10	.05	.01
SWTYPE	7.14	Yes	Yes	No
TESTER	1.32	No	No	No
SWTYPE*TESTER	1.32	No	No	No
SESSION	1.43	No	No	No
TECHNIQUE	137.55	Yes	Yes	Yes
RESIDUAL	1.18	No	No	No

Interpretation: The technique effect is highly significant and the application type is significant at the 95% level. This significance may be due to the code reading branch coverage values and needs to be explored further.

T TESTS OF PAIRED MEANS: Avg. PT_PATH

INTERPRETATIONS:

- SWTYPE** • SCENE Project has significantly lower averages. This observation may be related to complexity of the code samples used.
- TESTER** • T1 and T2 show significantly different average coverage. This observation may be related to code sample complexity.
- SESSION** • Not significantly different between sessions.
- TECHNIQUE** • Branch obtains significantly higher coverage. Code reading obtains significantly lower (zero) coverage.

5.2.2.4 Single Technique Effort Results

Refer also to SAS data in the Task 5 report.

Analysis Description from the SRTIP:

Identifier: TT_SINGLE_EFFORT_01

Question of Interest: Is there a difference in the effort (hours) required to meet the stopping rules by each test technique (R, F, B, C, S, E) at the (unit, CSC) test level?

Analysis: Analysis of Variance Model F Test

Interpretation of SAS output:

DEPENDENT VARIABLE: P_TTIME

FORMULA: P_TIME is technique time divided by sample time. Let technique time for the dynamic techniques be setup time + technique 'execution' time, where setup time is driver development; and DTM setup and static techniques have 0 setup time. Let sample time = $1 \times \text{setup time} + \text{each of the technique execution times for the 4 techniques in the latin square}$. (Note the sum of P_TIME s adds up to greater than 1 due to the addition of the setup time for each of the dynamic technique times. This quantity still reflects ACTUAL time to do things. Specifically, it reflects the time it would take to complete one technique in isolation (e.g., if you didn't already have a driver developed) divided by the actual time to complete the 4 sessions on the latin square for one sample.)

F TEST FOR MODEL SIGNIFICANCE:

F	α	Significance?
12.62	.10	Yes
	.05	Yes
	.01	Yes

INTERPRETATION: Model is highly significant.

F TEST FOR SOURCE SIGNIFICANCE:

SOURCE	F	Significance?		
		.10	.05	.01
SWTYPE	12.94	Yes	Yes	Yes
TESTER	1.75	No	No	No
SWTYPE*TESTER	0.68	No	No	No
SESSION	2.02	No	No	No
TECHNIQUE	47.1	Yes	Yes	Yes
RESIDUAL	0.76	No	No	No

INTERPRETATION: The technique effect is highly significant and the application type is significant at the 95% level.

T TESTS OF PAIRED MEANS: Ave. PT_TIME

INTERPRETATIONS:

- SWTYPE**
 - SCENE Project has significantly lower average. This observation may be related to complexity of the code samples used.
- TESTER**
 - T1 and T2 show significantly different average coverage. This observation may be related to code sample complexity.
- SESSION**
 - First session had significantly larger effort.
- TECHNIQUE**
 - Code Reading had significantly lower effort.

5.2.2.5 Single Technique Efficiency Results

Refer also to SAS data in the Task 5 report.

Analysis Description from the SRTIP:

Identifier: TT_SINGLE_EFFICIENCY_02

Question of Interest: Is there a difference in the effort (hours) required to detect a percent of total discrepancy reports by each test technique (R, F, B, C, S, E) at the unit test level?

Analysis: Analysis of Variance Model F Test

Interpretation of SAS output:

DEPENDENT VARIABLE: EFFIC

FORMULA: EFFIC IS SPRs found by a technique divided by technique time (P_TTIME).

F TEST FOR MODEL SIGNIFICANCE:

F	α	Significance?
2.48	.10	Yes
	.05	Yes
	.01	No

INTERPRETATION: Model is significant. Conduct all analyses at 95% confidence level or lower.

F TEST FOR SOURCE SIGNIFICANCE:

SOURCE	F	Significance?		
		.10	.05	.01
SWTYPE	.43	No	No	No
TESTER	3.47	Yes	Yes	No
SWTYPE*TESTER	.14	No	No	No
SESSION	.63	No	No	No
TECHNIQUE	7.32	Yes	Yes	Yes
RESIDUAL	1.27	No	No	No

INTERPRETATION: Both the tester and the technique have a significant effect on test efficiency. This result implies that we should concentrate on streamlining the techniques and on providing better training for testers.

T TESTS OF PAIRED MEANS: Ave. EFFIC

INTERPRETATIONS:

- SWTYPE • Not significantly different between projects.
- TESTER • T4 has significantly different effect on average efficiency than the other testers. This effect may be due to the complexity of the code samples tested by T4.
- SESSION • Not significantly different between sessions.
- TECHNIQUE • Code Reading had a significantly higher effect on efficiency.

5.2.3 General Observations

As noted in Section 5.1, while using these data, a few data discrepancies were found in the StatView data base, as well as between the StatView data base and the error summary tables provided in Section 5.2.6. Many were resolved. The project effort did not permit looking into the rest. While these discrepancies are believed to be minor, they also should be resolved before further analyses are performed.

The descriptive analyses present the data in a reduced form; they in essence are the first iteration in understanding the data and results. For example, the descriptive data showed it was possible that code review is the best technique to use at the unit level and that branch testing is the best technique to use at the CSC level.

The statistical analyses of variance, modeled according to the latin square experiment design, are the next iteration in understanding the data and results.

This report presented preliminary analysis of variance results for several performance measures - namely, P_SPR, PT_PATH, PT_TIME, and EFFICIENCY. A basic assumption underlying the analysis of variance model is that the error (deviations from the model) have a homogeneous variance. In particular, the magnitude of the error variance should not be dependent on the magnitude of the variate.

With a change in the PT_TIME variate (which represents percentage of total sample test time taken to setup and execute a given test technique) to a TTLTIME variate, where TTLTIME is defined as the time to setup and execute a test technique, this appears to be a reasonable assumption. However it is not reasonable for the other three dependent variables, for two reasons. First, the variates are percentages, and as such, the variation is dependent on their magnitude (i.e. variation tends to be smaller for percentages near 0 or 100 than those near 50 percent). Second, the base (i.e. denominator) for calculating the percentages (e.g. potential number of SPRs) varies drastically among the 32 cells, and the variance of the percentages is inversely related to these base numbers. The analysis of variance results shown herein for the two variables should therefore be interpreted with caution.

Categorical data analysis techniques offer a more appropriate approach for analysis of these type of performance measures, because these techniques properly account for the variance heterogeneity. The data for application of such a procedure can be regarded as a contingency table in which the 32 rows correspond to the 32 cells of the design and the columns correspond to a dichotomous response variable- e.g., SPRs found versus not found. The data are frequency counts. The proportions (analogous

to P_SPR), or some transformation of the proportions, can then be modeled as a function of the experiment design variables and tests of hypotheses concerning the effects can be performed. The estimation is usually performed by weighted least squares or by maximum likelihood techniques. (The latter would be preferred in this case due to the small counts.) In contrast to the F tests associated with the usual analysis of variance, the categorical data analysis approach employs a series of chi-square tests for the significance of the various design factors. Software for these analyses is available in SAS using the FREQ and more preferably the CATMOD procedures. Section 5.5 shows chi-square output using the FREQ procedure for test technique combinations. This output has not been reviewed for appropriateness and should also be interpreted with caution.

While carrying these analyses further is outside the scope of this contract, it is recommended that further work be done. In summary, complete and appropriate experimental data analysis often involves several iterative steps. Further analysis of these experiment results, under the guidance of a well-qualified statistician, is highly recommended.

5.2.4 SAS Outputs: GLM Results

Appendix A of the Task 5 report contains the output from the GENERAL LINEAR MODELS (GLM) procedure of the STATISTICAL ANALYSIS SYSTEM (SAS) software tool to analyze experiment data from the unit test level latin squares. Four GLM procedures were run on the following data sets: test technique effectiveness (SPRs), test technique effort, test technique branch coverage, and test technique efficiency.

5.2.5 SAS Output: CHI-SQUARE Results

Appendix B of the Task 5 report contains SAS outputs for the following four Chi-Square runs: unit effectiveness (SPRs), unit branch coverage, CSC effectiveness (SPRs), and CSC branch coverage. The outputs are available as a first iteration analysis for use in further work.

5.2.6 Error Summary Tables

The following error summary tables document all known unique errors in each sample, both for the unit and CSC testing. (A count of the rows in a table yields the total of all known unique errors for a sample.) The tables show which errors were found during experimental testing and by which test technique(s) (see the columns *B*, *C*, *F*, *R*, *EA*, *SA*), as well as showing which errors were from the original development testing. Only a portion of the original development SPRs (see column *Dev't SPR ID*) were determined to be findable at the unit and/or CSC levels; those errors are marked with a Y in the column *Dev't Findable SPR?*

The column labeled Error Source represents the testers and project consultants best estimate of where the error was introduced: for the dynamic techniques, failures were not traced to faults so the exact source of the error has not been verified.

5.3 RPFOM Exploratory Analysis

This section presents methods and results of exploratory analysis of selected RPFOM metrics components. Specifically, the relationships between software reliability and two RPFOM metrics, modularity and complexity, are investigated. These analyses focus upon the following questions of interest:

1. Is the occurrence of errors for a software component independent of the size of the component?
2. Is the occurrence of errors for a software component independent of the complexity of the component.

These questions were examined by applying simple linear regression to data compiled for the SCENE project. These data include error counts, size, and complexity obtained for individual units (i.e., subroutines) and combined for CSCs. A similar analysis of AFATDS SIM/STIM project data was not completed due to problems encountered which are detailed below. Regression analysis was conducted on a Macintosh using the Statview+ statistics package.

5.3.1 Data

Two Statview+ data files were constructed, one for unit-level analysis (Appendix C) and one for CSC-level analysis (Appendix D). Tables 5.23 and 5.24 provide descriptions of these data files.

Error, size (i.e., lines of code), and complexity values for units were available from existing SRM/TIT data sources described below. Remaining unit data, such as error density, were derived from the existing data. CSC data were derived from unit data: error, size, and complexity for CSCs represent sums of the values for the corresponding units.

5.3.1.1 Error Data

Error counts were compiled from the SCENE Error Density Matrix appearing in Appendix D of the SRM/TIT Task 2 Report, Volume 3. These error counts were originally logged from Software Problem Reports (SPRs) provided by SCENE project personnel. Each error for a unit represents a reference to that unit in a SPR. Each SPR may reference one or more units, and thus may account for multiple errors.

Two sets of error counts are recorded in the Statview unit and CSC data files. One set, labelled V6_ERRORS, represents SPRs logged against Version 6.0 (and versions thereafter) of SCENE; size and complexity metrics (see below) for units were extracted from Version 6.0 of the SCENE software. The other set, labelled ALL ERRORS, comprises all errors (including pre-Version 6.0 errors) from the SCENE Error Density Matrix for specified units and CSCs. Although pre-Version 6.0 errors for a software module cannot be directly associated with Version 6.0 metrics if the module has undergone change, this second set of error counts was also incorporated in analyses since it provided a larger sample size.

The organization of units into CSCs as represented in the Statview data files parallels that depicted in the SCENE Error Density Matrix. This scheme originates in documentation provided by SCENE project personnel (see Task 2 Report, Volume 3).

5.3.1.2 Metrics Data

Size and complexity measurements of SCENE units, collected automatically using the AMS tool, were imported into the Statview unit data file from the DBMS. Several functional definitions of these metrics are reported in the literature. For example, module size is defined variously as number of 1) lines of code, 2) executable lines of code, and 3) executable statements. These definitions themselves can be ambiguous. Since unambiguity and consistency in units of measurement are necessary for

Table 5.23 RPFOM Analysis Unit Data File Description

Field	Description
Unit	Unit Name
CSC	CSC Name
LOC	Unit Lines of Code from AMS
sx	Unit Complexity from AMS
All Errors	Total SPR Count for unit
V6_Errors	Vers. 6.0 or later SPR count for unit
All Errors/LOC	Error density (Errors/Lines of Code) for unit based on total SPR count
V6_Errors/LOC	Error density for unit based on SPRs for Version 6.0 or later of code

Table 5.24 RPFOM Analysis CSC Data File Description

Field	Description
CSC	CSC Name
LOC	Total lines of Code for all units of CSC
sx	Sum of unit complexities for CSC
All Errors	Total SPR counts for units of CSC
N_Units	Number of units (subroutines) in CSC
V6_Errors	SPR counts for units of CSC logged against Version 6.0 or later of code.
LOC/Unit	Mean lines of code for units of CSC
All Errors/LOC	Error density (Errors/Lines of Code) for CSC based on total SPR counts
sx/N_Units	Mean unit complexity for CSC
V6_Errors/LOC	Error density for CSC based on SPRs logged against Version 6.0 (or later vers.) of code

proper interpretation of results and for comparison of results with those of other studies, a precise functional definition of these metrics was sought.

The AMS tool reports module size as lines of code (LOC). Source code from four SCENE units was examined visually to establish the precise meaning of LOC as provided by AMS. The results indicate that each counted line of code, including each continuation line of a Fortran statement, increments the value of LOC by one. The only lines of code not counted are blank lines and Comment lines. The manner in which AMS treats Include statements is uncertain. These were converted to Comment lines for SCENE following questionable measurements obtained for AFATDS (see below), in which case very large values for LOC indicated that AMS may have counted lines of code in the Include files themselves.

The measure of complexity employed in the SRPEG for software units is the sum of conditional and unconditional branch statements. This measurement is one of several provided by AMS under headings of complexity and simplicity. Visual examination of the four SCENE units referenced above indicates that the AMS branch statement complexity number includes all IF and ELSE IF, DO and DO WHILE, GOTO, CALL, and RETURN statements.

5.3.1.3 Missing Data

Metrics data was unavailable for about 15 of the approximately 150 units listed in the SCENE Error Density Matrix. Four of these excluded units each represented an entire CSC. Difficulties encountered in collecting data with AMS is the primary reason for absence of this data.

Seventeen units for which AMS provided metrics data are missing from the SCENE Error Density Matrix, and consequently cannot be associated with any error counts. These units, included in the Statview database but excluded from the analyses, apparently were nonexistent when the SCENE project documentation (see Task 2 Report, Volume 3) outlining software components was produced.

The AFATDS SIM/STIM project was not included in these analyses due to spurious data from AMS on LOC and complexity (Table 5.25), lack of association between version of source code from which metrics were collected and versions against which SPRs were logged, and difficulties encountered when importing metrics data from the 4th Dimension RPFOM Database into Statview data files.

Table 5.25 presents LOC, unit complexity (sx), and CSC complexity (SX) values for AFATDS test sample units and CSCs (i.e., "integrated units"). High values for LOC and unit complexity (indicated in bold face) from late-build AFATDS source code, which served as the source of metrics data collection for all AFATDS units, raised suspicions concerning validity of AMS measurements. Since valid metrics data were necessary for test samples in order to complete analyses of results of the software testing experiment, AMS was run on the early-build versions of test sample units which were actually utilized in the testing experiment. Re-runs of AMS were also conducted on the late-build versions of three of the units. Based on the newly-generated metrics values, which are listed under "Sample Build," and visual examination of late-build versions of selected test-sample units, it was concluded that AMS had provided anomalous LOC and unit complexity values for an undetermined number of AFATDS units. The most likely explanation for this problem is that AMS processed INCLUDE files for units in which INCLUDE statements had not been converted to Comment lines.

Table 5.25. LOC & COMPLEXITY COMPARISONS BETWEEN
AFATDS SIM/STIM TEST SAMPLE BUILDS

Test Level	Unit/CSC Name	Lines of Code			Unit Complexity (SX)			Higher Level Complexity (SX)		
		Late Build	Sample Build	Late Build**	Late Build	Sample Build	Late Build**	Late Build	Sample Build	Late Build (Rev.)
Unit	SNWMSN*	36	45	36	133	10	10	1.500	1.000	1.000
	SMSMID	30	28	30	119	6	6			
	SINSEV	35	10		9	4				
	STUPMR	17	13		4	4				
	THUADS	386	49		18	8				
	THUADC*	501	29	52	18	6	18	1.000	0.800	1.000
	THUIEM	424	11		3	3				
	THPOON	584	130		35	33				
	THUSCN*	491	32		11	6		1.000	0.800	
	THUCOM	479	28		10	9				
	THURDT	576	80		31	31		1.000	1.000	
	THXRDT*	522	79		15	15				
	THUJWM	423	10		3	3				
CSC	SNWMSN*	118	96					1.200	0.850	?
	THPCON*	1978	201					1.075	1.025	?
	THUADS*	1311	89				N/A	0.933	0.867	?
	THURDT*	1945	180					1.025	1.025	?

* Test Samples

** AMS Re-runs

5.3.2 Analytical Approach

Simple linear regression was utilized to investigate relationships between software error occurrence as the dependent variable, and software component size and complexity as independent variables. Of interest is the degree to which results of these analyses support or refute the CSC/CSCI Modularity and Complexity models incorporated in the RPFOM computation. The equations representing these models are reviewed below (revisions made to original SRPEG values during the course of the SRM/TIT study are indicated in brackets):

$$SM \text{ (Modularity)} = (0.9u + w + 2x)/NM$$

where u = no. of units with $LOC < 200$ [100]
 w = no. of units with 200 [100] $< LOC < 3000$ [500]
 x = no. of units with $LOC > 3000$ [500]
 NM = no. of units

$$SX \text{ (Complexity)} = (1.5a + b + 0.8c)/NM$$

where a = no. of units with $sx > 20$
 b = no. of units with $7 < sx < 20$
 c = no. of units with $sx < 7$

The following null (H_0) hypotheses are tested:

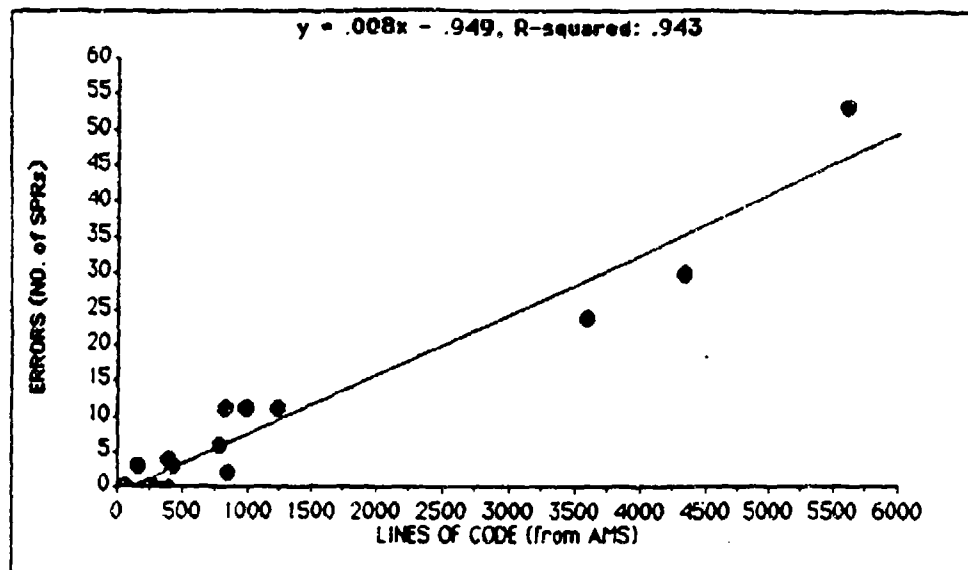
1. The number of software errors is independent of the size (measured by LOC) of a software component.
2. Error density is independent of the size (measured by LOC) of a software component.
3. The number of software errors is independent of the complexity (sx) of a software component.
4. Error density is independent of the complexity (sx) of a software component.

The F-test is employed to accept or reject these hypotheses at a .05 level of significance.

5.3.3 Results

5.3.3.1 Modularity

The graphical results for linear regression of errors (dependent variable) and lines of code (independent variable) for SCENE CSCs presented in Figure 5.11 (Version 6.0 errors used) and Figure 5.12 (total errors used) suggest a significant positive association between these two variables. High correlation coefficients ($R = .971$ in Fig. 5.11; $R = .968$ in Fig. 5.12) verify this association, and results of F-tests indicate a probability exceeding 99% that CSC error counts are dependent upon CSC size measured by lines of code ($p = .0001$ that the observed association between these variables is due to chance). Values for the coefficient of determination ($R^2 = .943$ in Fig. 5.11; $R^2 = .937$ in Fig. 5.12) demonstrate that a high proportion of the total variation in error count values is explained by the association of numbers of errors with lines of code.



Simple Regression X1: LOC Y1: V6_ERRORS				
DF:	R:	R-squared:	Adj. R-squared:	Std. Error:
16	.971	.943	.94	3.505

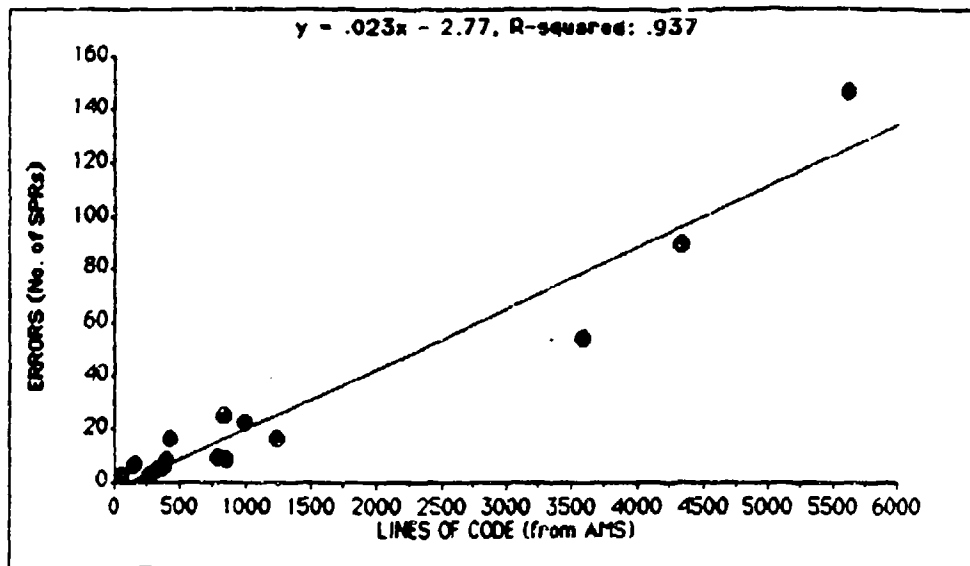
Analysis of Variance Table				
Source	DF:	Sum Squares:	Mean Square:	F-test:
REGRESSION	1	3069.296	3069.296	249.897
RESIDUAL	15	184.234	12.282	p = .0001
TOTAL	16	3253.529		

No Residual Statistics Computed

Simple Regression X1: LOC Y1: V6_ERRORS					
Beta Coefficient Table					
Parameter:	Value:	Std. Err.:	Std. Value:	t-Value:	Probability:
INTERCEPT	-.949				
SLOPE	.008	.001	.971	15.808	.0001

Confidence Intervals Table				
Parameter:	95% Lower:	95% Upper:	90% Lower:	90% Upper:
MEAN (X,Y)	7.482	11.106	7.804	10.784
SLOPE	.007	.01	.007	.009

Figure 5.11. SCENE CSC Regression of Errors (Version 6.0 SPRs) and Lines of Code.



Simple Regression X ₁ : LOC Y ₁ : ALL ERRORS				
DF:	R:	R-squared:	Adj. R-squared:	Std. Error:
16	.968	.937	.933	10.08

Analysis of Variance Table				
Source	DF:	Sum Squares:	Mean Square:	F-test:
REGRESSION	1	22656.925	22656.925	222.999
RESIDUAL	15	1524.016	101.601	p = .0001
TOTAL	16	24180.941		

No Residual Statistics Computed

Simple Regression X ₁ : LOC Y ₁ : ALL ERRORS					
Beta Coefficient Table					
Parameter:	Value:	Std. Err.:	Std. Value:	t-Value:	Probability:
INTERCEPT	-2.77				
SLOPE	.023	.002	.968	14.933	.0001

Confidence Intervals Table				
Parameter:	95% Lower:	95% Upper:	90% Lower:	90% Upper:
MEAN (X,Y)	19.848	30.27	20.773	29.345
SLOPE	.02	.026	.02	.026

Figure 3.12. SCENE CSC Regression of Errors (All SPRs) and Lines of Code.

Results of linear regression of these same variables for the unit components of the CSCs, illustrated in Figure 5.13 (Version 6.0 errors used) and Figure 5.14 (total errors used), support the findings presented above for CSCs. Although the correlation between error count and module size is weaker ($R = .78$ and $.81$) and less of the variation in error count is explained by the fitted regression ($R^2 = .61$ and $.66$), the probability of a dependent relationship of number of errors on lines of code remains greater than 99% ($p = .0001$ from F-test).

The positive correlation between error count and module size does not provide insight into the relationship between error frequency and module size. This was investigated by performing linear regression on errors per lines of code as the dependent variable, and lines of code as the independent variable. The results for CSCs presented in Figure 5.15 (Version 6.0 errors used) and Figure 5.16 (all errors used) indicate that error density cannot be predicted by CSC size ($p > .45$ that these variables are independent based on F-test). Similar results were obtained with error density and unit size, illustrated in Figure 5.17.

5.3.3.2 Complexity

The results of linear regression analysis of errors (dependent variable) and complexity (independent variable) for CSCs shown in Figure 5.18 (Version 6.0 errors used) and Figure 5.19 (all errors used) indicate a positive correlation between errors and complexity ($R > .90$ for both regressions). Results of F-tests imply a statistically significant dependence of error counts on complexity ($p = .0001$ that observed relationship is due to chance).

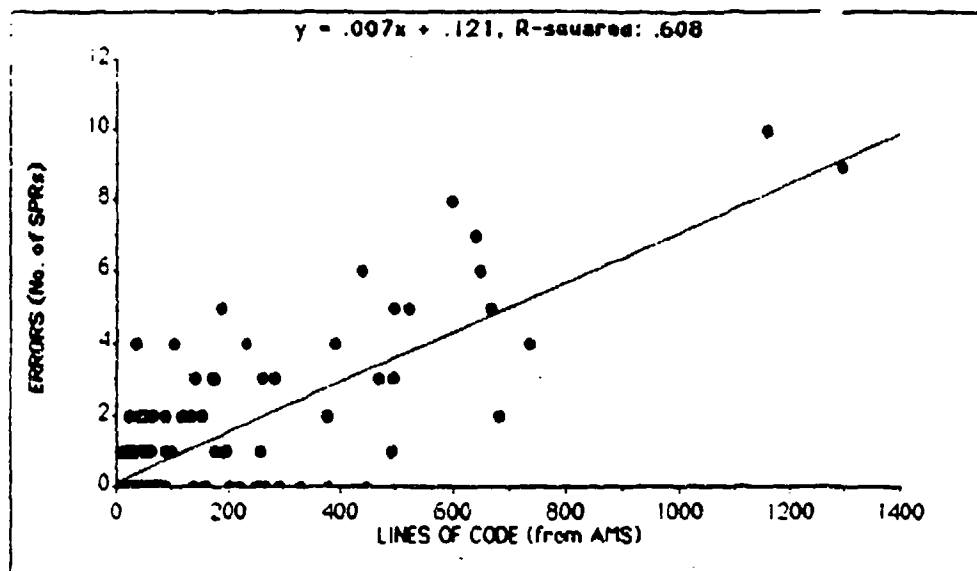
Analogous results were obtained when regressions were performed on error counts and complexity for units (Figures 5.20 and 5.21). However, as occurred in unit-level regressions of errors and module size described above, the correlations are not as high for units as for CSCs, and only half of the variation in error counts is attributable to its association with complexity ($R^2 < .50$).

To investigate the relationship of error frequency to complexity, linear regressions were conducted on error density (errors per lines of code) and complexity for CSCs (Figures 5.22 and 5.23) and units (Figures 5.24 and 5.25). The results indicate no significant relationship between error density and complexity except when error density for units was based on total error count (Figure 5.25). In this case, error density declined significantly with increased unit complexity according to F-test results ($p = .0111$ that this relationship is due to chance).

5.3.4 Summary and Conclusions

Results of linear regression analysis on SCENE units and CSCs indicate that occurrence of errors in software is not independent of module size or complexity. Specifically, it is shown that size of units or CSCs measured as lines of code, and complexity measured as number of branch statements, are determinants of error occurrence measured as number of Software Problem Reports (SPRs). These findings lead to rejection of null hypotheses #1 and #3 described in Section 5.3.6.2 above. That is, an increase in size or complexity of a software component (i.e., unit or CSC) will effect an increase in number of errors for that component.

Conversely, the results suggest that larger or more complex software units and CSCs do not experience a higher frequency of errors when frequency is measured as error density (errors per lines of code). In fact, when error density is based on total



Simple Regression X1: LOC Y1: V6_ERRORS

DF:	R:	R-squared:	Adj. R-squared:	Std. Error:
136	.78	.608	.605	1.242

Analysis of Variance Table

Source	DF	Sum Squares:	Mean Square:	F-test:
REGRESSION	1	323.564	323.564	209.772
RESIDUAL	135	208.232	1.542	p = .0001
TOTAL	136	531.796		

No Residual Statistics Computed

Simple Regression X1: LOC Y1: V6_ERRORS

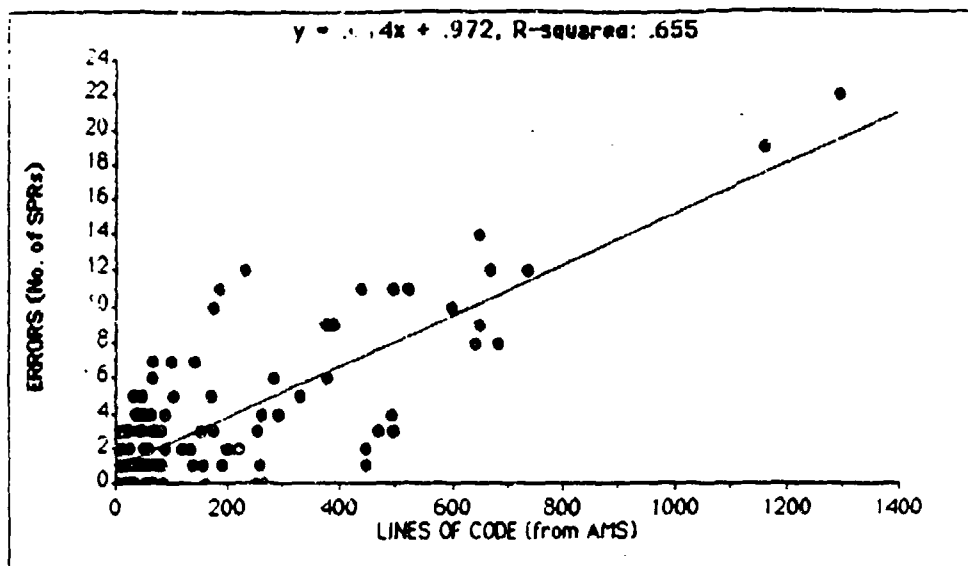
Beta Coefficient Table

Parameter:	Value:	Std. Err.:	Std. Value:	t-Value:	Probability:
INTERCEPT	.121				
SLOPE	.007	4.830E-4	.78	14.484	.0001

Confidence Intervals Table

Parameter:	95% Lower:	95% Upper:	90% Lower:	90% Upper:
MEAN (X,Y)	965	1.385	.999	1.351
SLOPE	.006	.008	.006	.008

Figure 5.13. SCENE Unit Regression of Errors (Version 6.0 SPRs and Lines of Code.



Simple Regression X1: LOC Y1: ALL ERRORS

DF	R	R-squared	Adj. R-squared	Std. Error
136	.809	.655	.652	2.294

Analysis of Variance Table

Source	DF	Sum Squares	Mean Square	F-Test
REGRESSION	1	1348.189	1348.189	256.093
RESIDUAL	135	710.701	5.264	p = .0001
TOTAL	136	2058.891		

No Residual Statistics Computed

Simple Regression X1: LOC Y1: ALL ERRORS

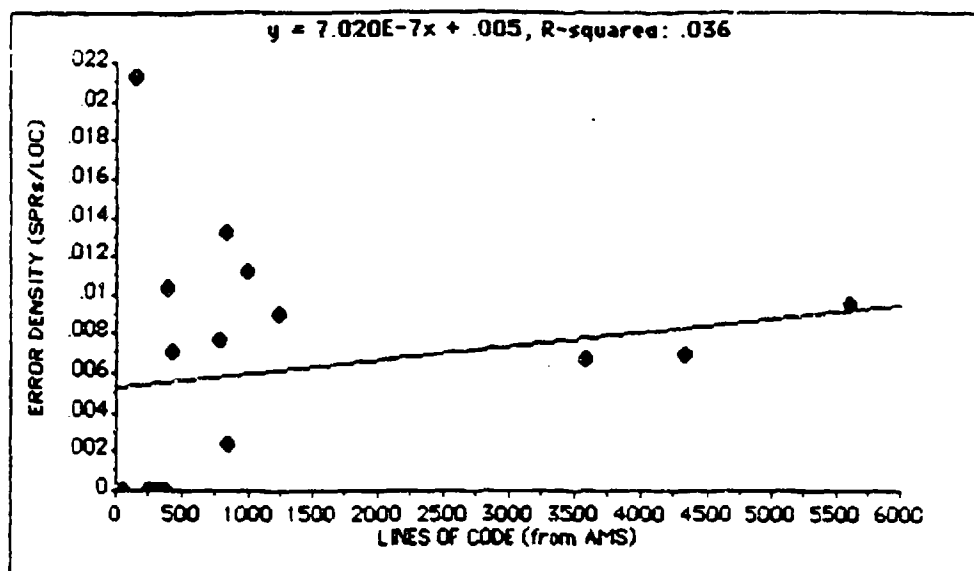
Beta Coefficient Table

Parameter	Value	Std. Err.	Std. Value	t-Value	Probability
INTERCEPT	.972				
SLOPE	.014	.001	.809	16.003	.0001

Confidence Intervals Table

Parameter	95% Lower	95% Upper	90% Lower	90% Upper
MEAN (X,Y)	2.736	3.512	2.799	3.449
SLOPE	.013	.016	.013	.016

Figure 5.14 SCENE Unit Regression of Errors (All SPRs) and Lines of Code.



Simple Regression X1: LOC Y1: V6_ERRORS/LOC

DF:	R:	R-squared:	Adj. R-squared:	Std. Error:
16	.191	.036	-.028	.006

Analysis of Variance Table

Source	DF	Sum Squares	Mean Square	F-test:
REGRESSION	1	2.128E-5	2.128E-5	566
RESIDUAL	15	.001	3.736E-5	p = .4633
TOTAL	16	.001		

No Residual Statistics Computed

Note: 1 case deleted with missing values.

Simple Regression X1: LOC Y1: V6_ERRORS/LOC

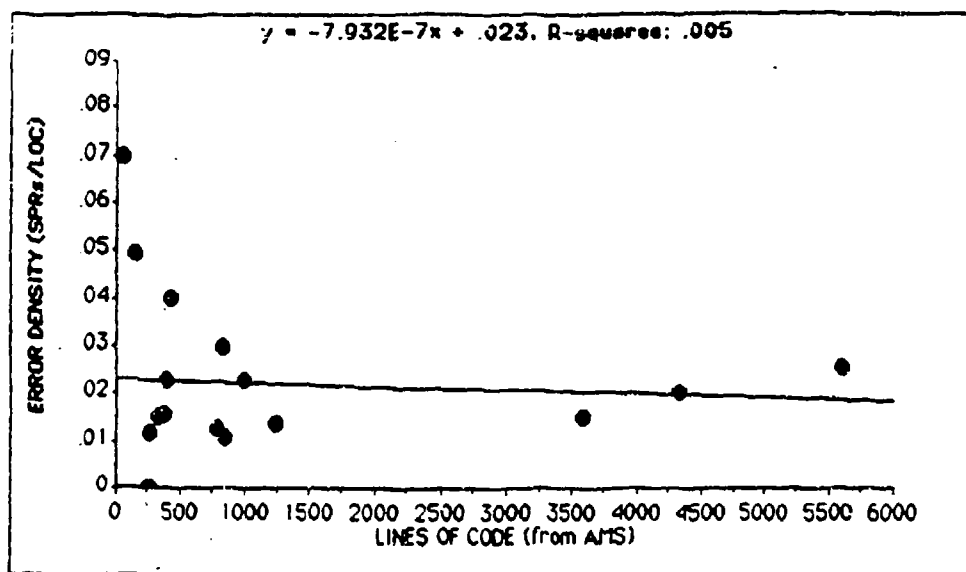
Beta Coefficient Table

Parameter:	Value:	Std. Err.:	Std. Value:	t-Value:	Probability:
INTERCEPT	.005				
SLOPE	7.020E-7	9.328E-7	.191	.753	.4633

Confidence Intervals Table

Parameter:	95% Lower:	95% Upper:	90% Lower:	90% Upper:
MEAN (X,Y)	.003	.009	.004	.009
SLOPE	-1.286E-6	2.691E-6	-9.334E-7	2.337E-6

Figure 5.15 SCENE CSC Regression of Error Density (based on Version 6.0 SPRs) and Lines of Code.



Simple Regression X1: LOC Y1: ALL ERRORS/LOC				
DF:	R:	R-squared:	Adj. R-squared:	Std. Error:
16	.074	.005	-.061	.018

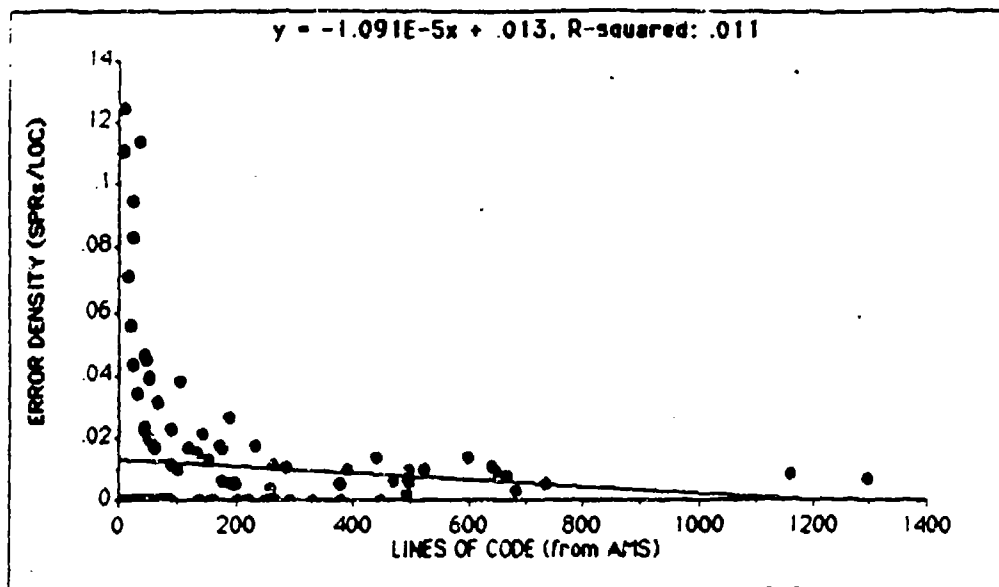
Analysis of Variance Table				
Source	DF:	Sum Squares:	Mean Square:	F-test:
REGRESSION	1	2.716E-5	2.716E-5	.083
RESIDUAL	15	.005	3.276E-4	p = .7773
TOTAL	16	.005		

No Residual Statistics Computed

Simple Regression X1: LOC Y1: ALL ERRORS/LOC					
Beta Coefficient Table					
Parameter:	Value:	Std. Err.:	Std. Value:	t-Value:	Probability:
INTERCEPT	.023				
SLOPE	-7.932E-7	2.755E-6	-.074	.288	.7773

Confidence Intervals Table				
Parameter:	95% Lower:	95% Upper:	90% Lower:	90% Upper:
MEAN (X,Y)	.013	.032	.015	.03
SLOPE	-6.666E-6	5.079E-6	-5.623E-6	4.037E-6

Figure 5.16 SCENE CSC Regression of Error Density (based on all SPRs) and Lines of Code.



Simple Regression X1: LOC Y1: V6_ERRORS/LOC				
DF:	R:	R-squared:	Adj. R-squared:	Std. Error:
136	.103	.011	.003	.023

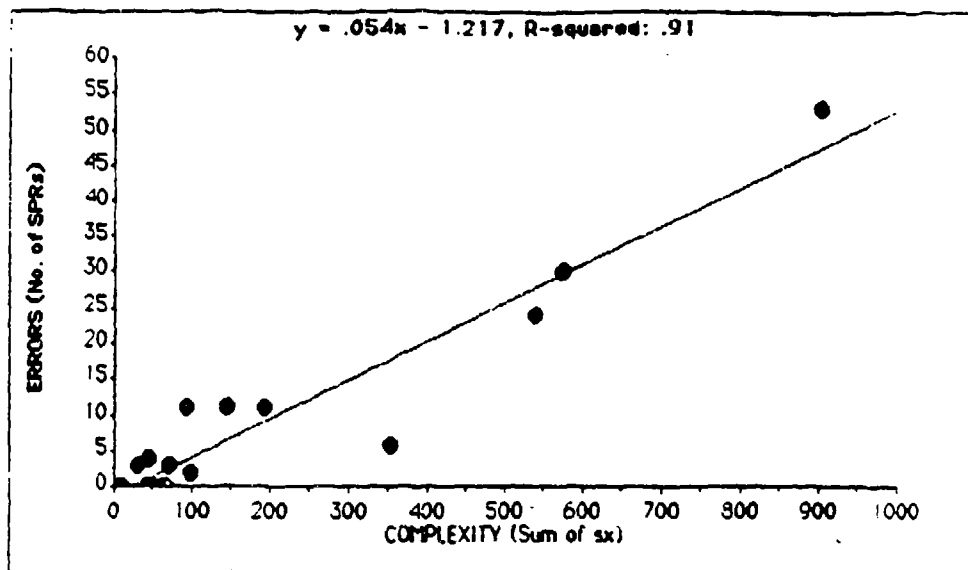
Analysis of Variance Table				
Source	DF:	Sum Squares	Mean Square	F-test:
REGRESSION	1	.001	.001	1.458
RESIDUAL	135	.073	.001	p = .2294
TOTAL	136	.074		

No Residual Statistics Computed

Simple Regression X1: LOC Y1: V6_ERRORS/LOC					
Beta Coefficient Table					
Parameter:	Value:	Std. Err.:	Std. Value:	t-Value:	Probability:
INTERCEPT	.013				
SLOPE	-1.091E-5	9.034E-6	-.103	1.207	.2294

Confidence Intervals Table				
Parameter:	95% Lower:	95% Upper:	90% Lower:	90% Upper:
MEAN (X,Y)	.007	.015	.008	.015
SLOPE	-2.878E-5	6.960E-6	-2.587E-5	4.056E-6

Figure 5.17 SCENE Unit Regression of Error Density (based on Version 6.0 SPRs) and Lines of Code



Simple Regression: X1: sx Y1: V6_ERRORS

DF:	R	R-squared:	Adj. R-squared:	Std. Error:
16	.954	.91	.904	4.427

Analysis of Variance Table

Source	DF:	Sum Squares:	Mean Square:	F-test:
REGRESSION	1	2959.494	2959.494	150.976
RESIDUAL	15	294.036	19.602	p = .0001
TOTAL	16	3253.529		

No Residual Statistics Computed

Simple Regression X1: sx Y1: V6_ERRORS

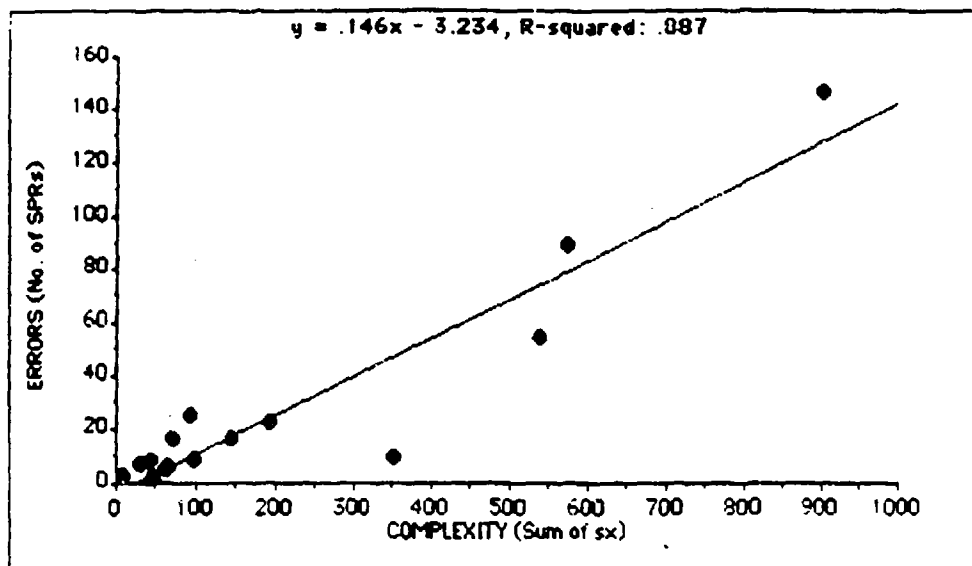
Beta Coefficient Table

Parameter:	Value:	Std. Err.:	Std. Value:	t-Value:	Probability:
INTERCEPT	-1.217				
SLOPE	.054	.004	.954	12.287	.0001

Confidence Intervals Table

Parameter:	95% Lower:	95% Upper:	90% Lower:	90% Upper:
MEAN (X,Y)	7.005	11.583	7.411	11.177
SLOPE	.045	.063	.046	.062

Figure 5.18 SCENE CSC Regression of Errors (Version 6.0 SPRs) and Complexity (sum for units).



Simple Regression X_1 : sx Y_1 : ALL ERRORS

DF:	R:	R-squared:	Adj. R-squared:	Std. Error:
16	.942	.887	.879	13.515

Analysis of Variance Table

Source	DF:	Sum Squares:	Mean Square:	F-test:
REGRESSION	1	21441.257	21441.257	117.393
RESIDUAL	15	2739.684	182.646	p = .0001
TOTAL	16	24180.941		

No Residual Statistics Computed

Note: 1 case deleted with missing values.

Simple Regression X_1 : sx Y_1 : ALL ERRORS

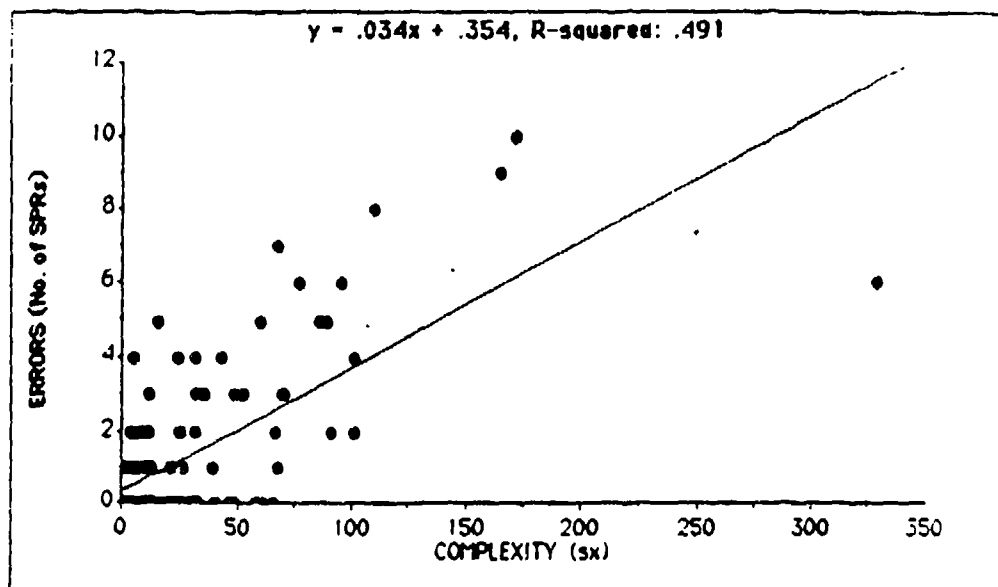
Beta Coefficient Table

Parameter:	Value:	Std. Err.:	Std. Value:	t-Value:	Probability:
INTERCEPT	-3.234				
SLOPE	.146	.013	.942	10.835	.0001

Confidence Intervals Table

Parameter:	95% Lower:	95% Upper:	90% Lower:	90% Upper:
MEAN (X,Y)	18.072	32.046	19.312	30.806
SLOPE	.117	.174	.122	.169

Figure 5.19 SCENE CSC Regression of Errors (all SPRs) and Complexity (sum for units).



Simple Regression X1: sx Y1: V6_ERRORS

DF:	R:	R-squared:	Adj. R-squared:	Std. Error:
136	.701	.491	.487	1.416

Analysis of Variance Table

Source	DF:	Sum Squares:	Mean Square:	F-test:
REGRESSION	1	261.147	261.147	130.26
RESIDUAL	135	270.649	2.005	p = .0001
TOTAL	136	531.796		

No Residual Statistics Computed

Simple Regression X1: sx Y1: V6_ERRORS

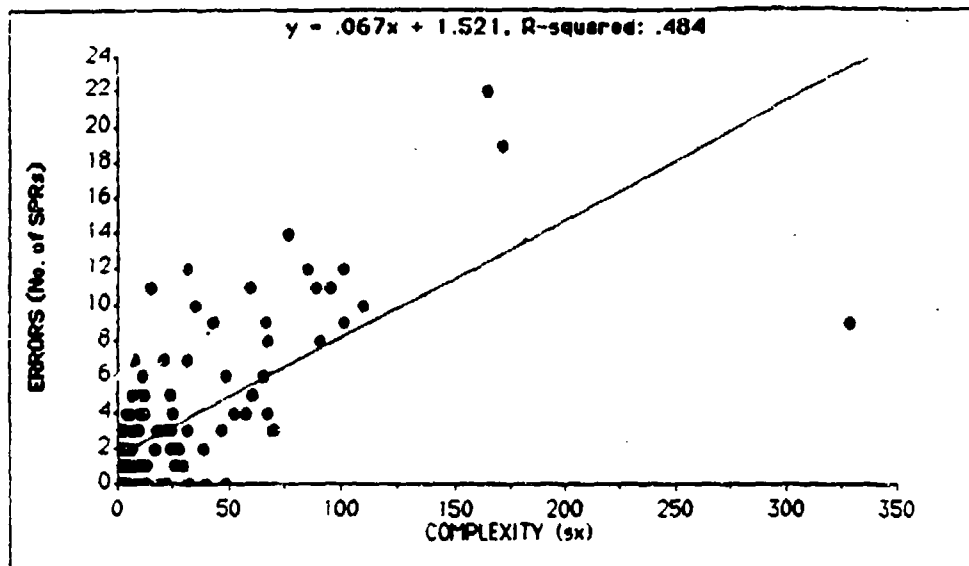
Beta Coefficient Table

Parameter:	Value:	Std. Err.:	Std. Value:	t-value:	Probability:
INTERCEPT	.354				
SLOPE	.034	.003	.701	11.413	.0001

Confidence Intervals Table

Parameter:	95% Lower:	95% Upper:	90% Lower:	90% Upper:
MEAN (X,Y)	.936	1.414	.975	1.376
SLOPE	.028	.04	.029	.039

Figure 5.20 SCENE Unit Regression of Errors (Version 6.0 SPRs and Complexity.



Simple Regression X1: sx Y1: ALL ERRORS

DF:	R:	R-squared:	Adj. R-squared:	Std. Error:
136	.696	.484	.48	2.805

Analysis of Variance Table

Source	DF:	Sum Squares:	Me. Square:	F-test:
REGRESSION	1	996.454	996.454	126.616
RESIDUAL	135	1062.437	7.87	p = .0001
TOTAL	136	2058.891		

No Residual Statistics Computed

Simple Regression X1: sx Y1: ALL ERRORS

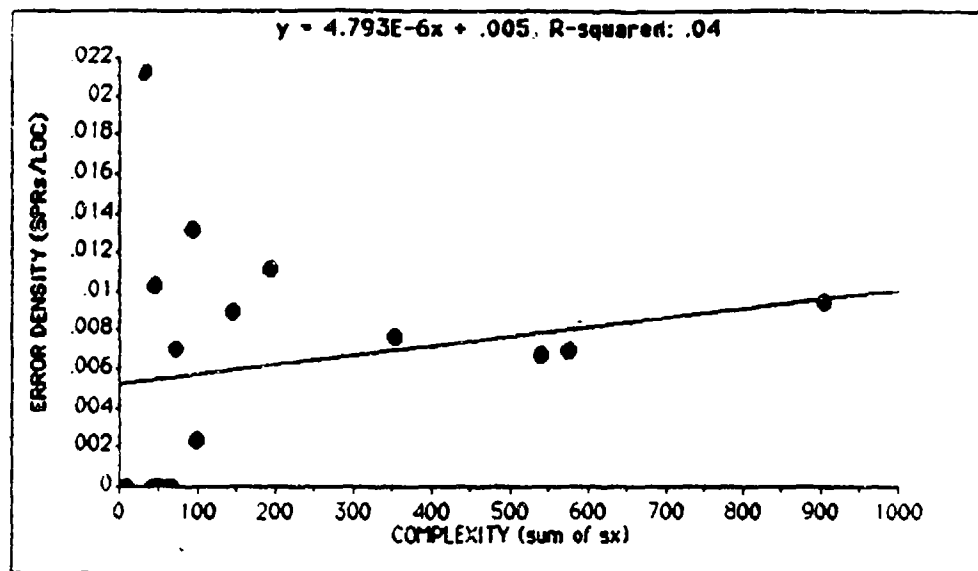
Beta Coefficient Table

Parameter:	Value:	Std. Err.:	Std. Value:	t-Value:	Probability:
INTERCEPT	1.521				
SLOPE	.067	.006	.696	11.252	.0001

Confidence Intervals Table

Parameter:	95% Lower:	95% Upper:	90% Lower:	90% Upper:
MEAN (X,Y)	2.65	3.598	2.727	3.521
SLOPE	.055	.078	.057	.076

Figure 5.21 SCENE Unit Regression of Errors (all SPRs) and Complexity.



Simple Regression X1: sx Y1: V6_ERRORS/LOC

DF:	R:	R-squared:	Adj. R-squared:	Std. Error:
16	.199	.04	-.024	.006

Analysis of Variance Table

Source	DF:	Sum Squares:	Mean Square:	F-test:
REGRESSION	1	2.323E-5	2.323E-5	.621
RESIDUAL	15	.001	3.743E-5	p = .4431
TOTAL	16	.001		

No Residual Statistics Computed

Simple Regression X1: sx Y1: V6_ERRORS/LOC

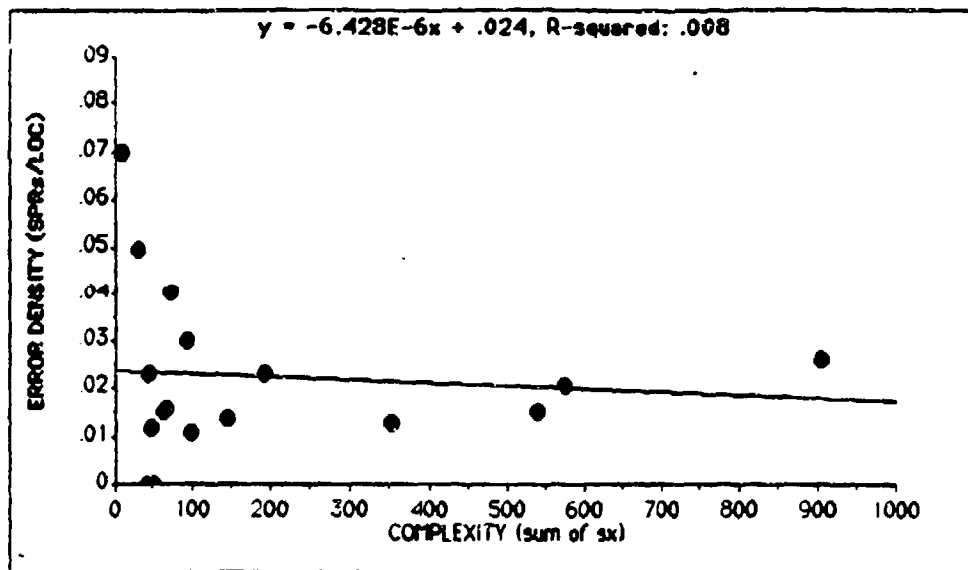
Beta Coefficient Table

Parameter:	Value:	Std. Err.:	Std. Value:	t-Value:	Probability
INTERCEPT	.005				
SLOPE	4.793E-6	6.084E-6	.199	.788	.4431

Confidence Intervals Table

Parameter:	95% Lower:	95% Upper:	90% Lower:	90% Upper:
MEAN (X,Y)	.003	.009	.004	.009
SLOPE	-8.177E-6	1.776E-5	-5.874E-6	1.546E-5

Figure 5.22 SCENE CSC Regression of Error Density (based on Version 6.0 SPRs) and Complexity (sum for units).



Simple Regression X₁: sx Y₁: ALL ERRORS/LOC

DF:	R:	R-squared:	Adj. R-squared:	Std. Error:
16	.092	.008	-.058	.018

Analysis of Variance Table

Source	DF:	Sum Squares:	Mean Square:	F-test:
REGRESSION	1	4.178E-5	4.178E-5	.128
RESIDUAL	15	.005	3.267E-4	p = .7256
TOTAL	16	.005		

No Residual Statistics Computed

1

Simple Regression X₁: sx Y₁: ALL ERRORS/LOC

Beta Coefficient Table

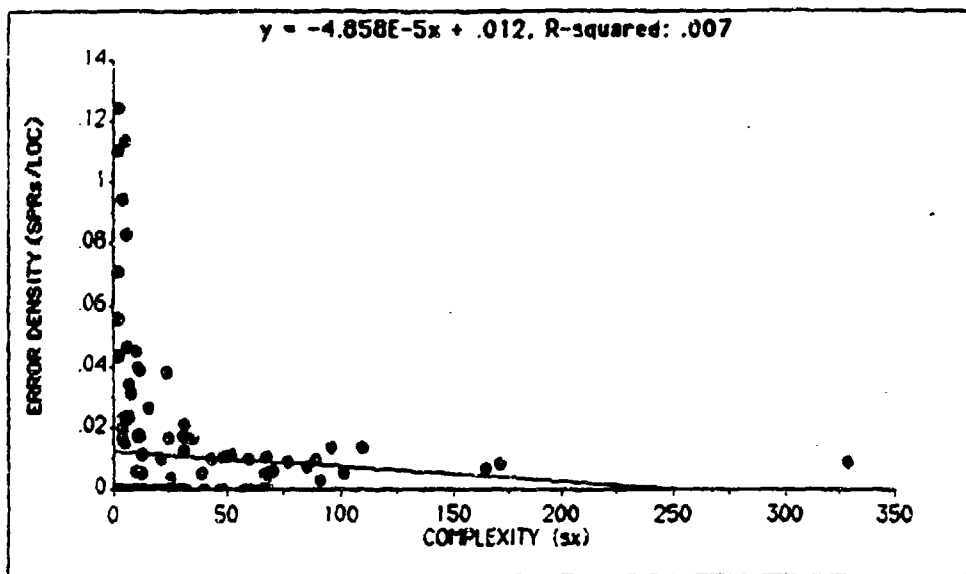
Parameter:	Value:	Std. Err.:	Std. Value:	t-Value:	Probability:
INTERCEPT	.024				
SLOPE	-6.428E-6	1.797E-5	-.092	.358	.7256

Confidence Intervals Table

Parameter:	95% Lower:	95% Upper:	90% Lower:	90% Upper:
MEAN (X,Y)	.013	.032	.015	.03
SLOPE	-4.474E-5	3.189E-5	-3.794E-5	2.508E-5

2

Figure 5.23 SCENE CSC Regression of Error Density (based on all SPRs) and Complexity (sum for units).



Simple Regression X1: sx Y1: V6_ERRORS/LOC

DF:	R:	R-squared:	Adj. R-squared:	Std. Error:
136	.085	.007	-1.327E-4	.023

Analysis of Variance Table

Source	DF:	Sum Squares:	Mean Square:	F-test:
REGRESSION	1	.001	.001	.982
RESIDUAL	135	.073	.001	p = .3235
TOTAL	136	.074		

No Residual Statistics Computed

1

Simple Regression X1: sx Y1: V6_ERRORS/LOC

Beta Coefficient Table

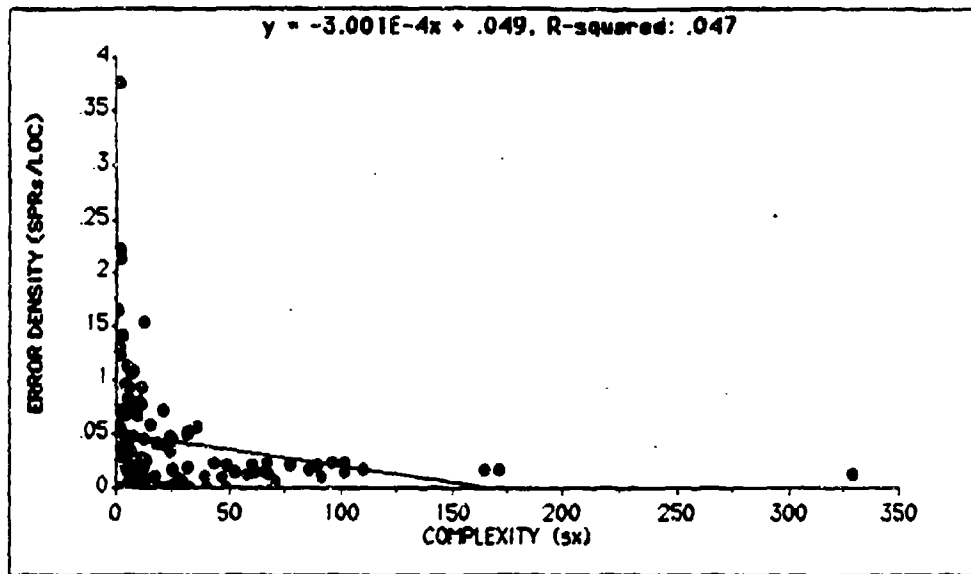
Parameter:	Value:	Std. Err.:	Std. Value:	t-Value:	Probability:
INTERCEPT	.012				
SLOPE	-4.858E-5	4.902E-5	-.085	.991	.3235

Confidence Intervals Table

Parameter:	95% Lower:	95% Upper:	90% Lower:	90% Upper:
MEAN (X,Y)	.007	.015	.006	.015
SLOPE	-1.455E-4	4.838E-5	-1.298E-4	3.262E-5

2

Figure 5.24 SCENE Unit Regression of Error Density (based on Version 6.0 SPRs) and Complexity.



Simple Regression X1: sx Y1: ALL ERRORS/LOC

DF:	R:	R-squared:	Adj. R-squared:	Std. Error:
136	.216	.047	.04	.055

Analysis of Variance Table

Source	DF:	Sum Squares:	Mean Square:	F-test:
REGRESSION	1	.02	.02	6.634
RESIDUAL	135	.413	.003	p = .0111
TOTAL	136	.433		

No Residual Statistics Computed

1

Simple Regression X1: sx Y1: ALL ERRORS/LOC

Beta Coefficient Table

Parameter:	Value:	Std. Err.:	Std. Value:	t-Value:	Probability:
INTERCEPT	.049				
SLOPE	-3.001E-4	1.165E-4	-.216	2.576	.0111

Confidence Intervals Table

Parameter:	95% Lower:	95% Upper:	90% Lower:	90% Upper:
MEAN (X,Y)	.033	.052	.034	.05
SLOPE	-.001	-6.964E-5	-4.931E-4	-1.071E-4

2

Figure 5.25 SCENE Unit Regression of Error Density (based on all SPRs) and Complexity.

errors (in contrast to subset of Version 6.0 errors) for units, a statistically significant association between declining error density and increasing complexity is shown based upon F-test results (Figure 5.25). However, in view of low correlation ($R = .216$) and determination ($R^2 = .047$) coefficients (Figure 5.25), and the absence of statistical significance when error density is based on Version 6.0 errors alone (Figure 5.24), it is concluded that significantly higher error density cannot be meaningfully correlated with lower complexity. These results of regression analyses on error density, and module size and complexity lead to acceptance of null hypotheses #2 and #4 proposed in Section 5.3.6.2. Specifically, error density is independent of software component size and complexity.

The results presented here are consistent with those from analyses conducted on other software development projects and reported in Volume 1 of the SRPEG. The relationship of these results to the concepts of modularity and complexity is open to interpretation. If "increase in errors" is defined as greater number of errors, then increasing the size or complexity of modules can be expected to result in an increase in errors per module. If, on the other hand, "increase in errors" represents greater error frequency measured as error density, then larger or more complex modules are not expected to experience an increase in errors.

From a software project management perspective, a system comprising 100,000 lines of code with 10,000 branch statements distributed among 1000 modules can be expected to experience the same number of errors as would an equivalent system distributed among 100 modules, all other factors remaining equal. In this simplistic example, a reduction in module size and complexity will only result in a reduction in errors if the frame of reference is the individual module (unit, CSC, etc.). A reduction in errors for the overall system could be achieved if total lines of code or total number of branch statements could be reduced without sacrificing functional capabilities. These observations do not support the modularity and complexity models presented in Section 5.3.2 since, instead of considering size and complexity of a CSC or CSCI, these models are based on numbers of component units exhibiting certain size and complexity characteristics. It is recommended that new measures of module size and complexity be investigated, and that new models relating software reliability to these metrics be derived from these new measures.

APPENDIX C

Unit-level Analysis Data File

	UNIT	CSC	LOC	SH	ALL ERRORS	U6_ERRORS	ALL ERRORS/LOC
1	USINASAB	CSINASAB	252	46	3	0	.012
2	USINEDENT	CSINEDENT	56	24	2	0	.036
3	USEDSDRY	CSINEDENT	75	22	3	0	.040
4	USEUUSACH	CSINEDENT	1295	165	22	9	.017
5	USINP_BMLN	CSINEDENT	104	24	5	4	.048
6	USINP_SHIL	CSINEDENT	158	32	0	0	0
7	USSHTL..IN	CSINEDENT	221	27	2	0	.009
8	USINP_NFL	CSINEDENT	495	69	3	3	.006
9	USINP_POI	CSINEDENT	65	12	0	0	0
10	USEDUMRY	CSINEDENT	88	25	4	0	.045
11	USINP_ASAT	CSINEDENT	468	70	3	3	.006
12	USDIRECT	CSINEDENT	62	8	0	0	0
13	USEDSDPLY	CSINEDENT	497	60	11	5	.022
14	USINFANSEN	CSINFANSEN	328	61	5	0	.015
15	USTARGCHK	CSINFIL	29	7	1	0	.034
16	USINFIL	CSINFIL	438	96	11	6	.025
17	USEDINFIL	CSINFIL	522	89	11	5	.021
18	USINFLAGS	CSINFLAGS	141	31	7	3	.050
19	USOUTENG	CSINITEUT	73	18	3	0	.041
20	USOUTDET	CSINITEUT	56	13	0	0	0
21	USOUTCON	CSINITEUT	46	9	1	0	.022
22	USOUTPOS	CSINITEUT	150	31	3	2	.020
23	USOUTFLU	CSINITEUT	83	19	0	0	0
24	USOUTEDE	CSINITEUT	667	85	12	5	.018
25	USNDRMAL	CSINITEUT	15	3	1	0	.067
26	USLAZMTH	CSINITEUT	51	11	4	2	.078
27	USLATLON	CSINITEUT	9	2	2	1	.222
28	USINITSAT	CSINITEUT	32	1	1	0	.031
29	USMTIMPY	CSINITEUT	446	4	1	0	.002
30	USMTIMPT	CSINITEUT	14	4	1	0	.071

	U6_ERRORS/LOC
1	0
2	0
3	0
4	.007
5	.038
6	0
7	0
8	.006
9	0
10	0
11	.006
12	0
13	.010
14	0
15	0
16	.014
17	.010
18	.021
19	0
20	0
21	0
22	.013
23	0
24	.007
25	0
26	.039
27	.111
28	0
29	0
30	0

	UNIT	CSC	LOC	SR	ALL ERRORS	U6_ERRORS	ALL ERRORS/LOC
31	USLUNDEC	CSINITEUT	51	5	2	0	.039
32	USOUTPUT	CSINITEUT	32	12	5	0	.156
33	USDEC SMP	CSINITEUT	7	2	1	0	.143
34	USVIEW	CSINITEUT	64	11	6	0	.094
35	USSVIEW	CSINITEUT	86	12	4	1	.047
36	USTRADEL	CSINITEUT	18	2	1	1	.056
37	USTRADE	CSINITEUT	42	7	3	1	.071
38	USDECROT	CSINITEUT	12	5	1	0	.083
39	USSUNDEC	CSINITEUT	43	6	4	2	.093
40	USADINTERP	CSINITEUT	13	2	0	0	0
41	USPOLYGEN	CSINITEUT	46	6	1	0	.022
42	USOUTSBS	CSINITEUT	56	6	1	0	.018
43	USSTGORB	CSINITEUT	56	10	1	1	.018
44	USSOUND	CSINITEUT	28	22	0	0	0
45	USSATPOS	CSINITEUT	596	110	10	8	.017
46	USINITIAL	CSINITEUT	231	31	12	4	.052
47	USB00STPH2	CSINITEUT	43	4	1	1	.023
48	USB00STPH	CSINITEUT	64	8	7	2	.109
49	USBSTATE	CSINITEUT	118	25	2	2	.017
50	USCONDTIME	CSINITEUT	16	0	1	0	.062
51	USCONPROC	CSINITEUT	57	7	2	0	.035
	USB00STINT	CSINITEUT	24	5	•	•	•
53	USALPROC	CSINITEUT	76	11	1	0	.013
54	USADID	CSINITEUT	5	3	0	0	0
55	USASATPRC	CSINITEUT	175	35	10	3	.057
56	USBMBOOST	CSINITEUT	35	5	4	4	.114
57	USASINI	CSINITEUT	9	2	0	0	0
58	USGROPOS	CSINITEUT	37	13	1	6	.027
59	USGPOSHY	CSINITEUT	7	1	0	0	0
60	USUALT	CSINITEUT	138	29	1	0	.007

	U6_ERRORS/LCC
31	0
32	0
33	0
34	0
35	.012
36	.056
37	.024
38	0
39	.047
40	0
41	0
42	0
43	.018
44	0
45	.013
46	.017
47	.023
48	.031
49	.017
50	0
51	0
	*
53	0
54	0
55	.017
56	.114
57	0
58	0
59	0
60	0

	UNIT	CSC	LOC	SN	ALL ERRORS	UG_ERRORS	ALL ERRORS/LOC
61	USINITFAN	CSINITIEDT	49	12	1	0	.020
62	USINITIUT	CSINITIUT	98	21	7	1	.071
63	USGAUSS2	CSINITIEDT	47	11	1	0	.021
64	USDENSITY	CSINITIUT	54	31	0	0	0
65	USCOORDB2	CSINITIEDT	282	48	6	3	.021
66	USEDUPROC	CSINITIUT	1157	171	19	10	.016
67	USGAUSS1	CSINITIUT	45	11	1	0	.022
68	USFLYOUT	CSINITIEDT	67	6	1	0	.015
69	USDJULA	CSINITIUT	9	1	2	0	.222
70	USALPHG	CSINMODE	23	2	3	1	.130
71	USINMODE	CSINMODE	376	66	9	2	.024
72	USBOV	CSINMODE	14	2	3	0	.214
73	USINOWNOP	CSINOWNOP	265	48	0	0	0
74	USINPALA	CSINPALA	250	40	0	0	0
75	USSORTSAT	CSINPSAT	156	7	1	0	.006
76	USINPSAT	CSINPSAT	680	91	8	2	.012
77	USINPUTSBS	CSINPUTSBS	490	67	4	1	.008
78	USINPUTESBS	CSINPUTSBS	640	67	8	7	.013
79	USSMA2MEAN	CSINPUTSBS	14	0	1	1	.071
80	USTAU2MEAN	CSINPUTSBS	21	4	2	2	.095
81	USSORTSBS	CSINPUTSBS	60	4	4	1	.067
82	USINSENS	CSINSENS	379	65	6	0	.016
83	USEDOUTPT	CSOUTFIL	646	77	14	6	.022
84	USOUTFIL	CSOUTFIL	186	15	11	5	.059
85	USSHTDWN	CSHTDWN	43	9	3	0	.070
	USINEPH	CSUNKNOWN	47	4	0	0	0
	USMNU-INSE	CSUNKNOWN	487	80	0	0	0
	USGD2EC1	CSUNKNOWN	31	1	0	0	0
	USFLINE	CSUNKNOWN	19	4	0	0	0
	USGAUSSH	CSUNKNOWN	54	12	0	0	0

	U6_ERRORS/LOC
61	0
62	.010
63	0
64	0
65	.011
66	.009
67	0
68	0
69	0
70	.043
71	.005
72	0
73	0
74	0
75	0
76	.003
77	.002
78	.011
79	.071
80	.095
81	.017
82	0
83	.009
84	.027
85	0
	•
	•
	•
	•
	•

	UNIT	CSC	LOC	SH	ALL ERRORS	U6_ERRORS	ALL ERRORS/LOC
	USSCATHSG	CSUNKNOON	10	3	•	•	•
	USTREC1	CSUNKNOON	17	4	•	•	•
	USPLANAR_H	CSUNKNOON	192	43	•	•	•
	USSPARE	CSUNKNOON	62	13	•	•	•
	USOBS	CSUNKNOON	21	42	•	•	•
	USEC12POL	CSUNKNOON	26	1	•	•	•
	USBOOSTPH_	CSUNKNOON	40	8	•	•	•
	USBAFFLE	CSUNKNOON	30	4	•	•	•
	USCALCAZ	CSUNKNOON	29	8	•	•	•
	USDELTAJD	CSUNKNOON	15	0	•	•	•
	USDECAY	CSUNKNOON	46	16	•	•	•
102	USPSUHAN	CSUTILITIES	133	5	2	2	.015
103	USORBITINIT	CSUTILITIES	80	9	1	0	.013
104	USNOBLANK	CSUTILITIES	17	4	0	0	0
105	USRESET	CSUTILITIES	196	39	2	1	.010
106	USREALIN	CSUTILITIES	50	10	4	2	.080
107	USREALCHC	CSUTILITIES	44	9	3	2	.068
108	USINTCHG	CSUTILITIES	29	•	1	1	.034
109	USINSERT	CSUTILITIES	81	24	3	0	.037
110	USHDRIER	CSUTILITIES	88	5	2	2	.023
111	USNEWSAT	CSUTILITIES	446	5	2	0	.004
112	USKEPLER	CSUTILITIES	47	7	5	0	.106
113	USINTIN	CSUTILITIES	24	6	2	2	.083
114	USUNORM	CSUTILITIES	8	2	1	0	.125
115	USHPROD	CSUTILITIES	19	2	0	0	0
116	USTAARGUAL	CSUTILITIES	30	5	0	0	0
117	USDCROSS	CSUTILITIES	8	1	1	0	.125
118	USUDOT	CSUTILITIES	6	1	1	0	.167
119	USUMAG	CSUTILITIES	6	1	1	0	.167
120	USSGPORB	CSUTILITIES	174	9	3	1	.017

	UNIT	CSC	LOC	SH	ALL ERRORS	U6_ERRORS	ALL ERRORS/LOC
121	USSCANNER	CSUTILITIES	292	58	4	0	.014
122	USRMAG	CSUTILITIES	6	1	0	0	0
123	USTARGET	CSUTILITIES	27	5	1	0	.037
124	USSATL_CHG	CSUTILITIES	200	17	2	0	.010
125	USSGPDIN	CSUTILITIES	171	11	5	3	.029
126	USCORRECT	CSUTILITIES	18	1	3	0	.167
127	USCONTACT	CSUTILITIES	377	101	9	2	.024
128	USCS	CSUTILITIES	21	3	3	0	.143
129	USCROSS	CSUTILITIES	8	1	0	0	0
130	USCNDTIM	CSUTILITIES	30	1	1	0	.033
131	USCHECK	CSUTILITIES	14	3	0	0	0
132	USACOS1	CSUTILITIES	9	2	0	0	0
133	USCNDEUT	CSUTILITIES	63	8	3	0	.048
134	USCLASCRN	CSUTILITIES	17	1	0	0	0
135	USDFMOD2P	CSUTILITIES	14	2	1	1	.071
136	USEDVIEW	CSUTILITIES	261	52	4	3	.015
137	USEUCOMPR	CSUTILITIES	735	101	12	4	.016
138	USHANDE	CSUTILITIES	256	26	1	1	.004
139	USFMOD2P	CSUTILITIES	8	2	3	1	.375
140	USELLNFL	CSUTILITIES	42	5	1	1	.024
141	USDPPER	CSUTILITIES	191	12	1	1	.005
142	USDOT	CSUTILITIES	6	1	0	0	0
143	USECIUDD	CSUTILITIES	26	3	1	0	.038
144	USECI26D	CSUTILITIES	51	4	1	1	.020
145	USCHKINT	CSVERIFY	13	2	0	0	0
146	USCHKILIS	CSVERIFY	18	4	0	0	0
147	USCHKLNKS	CSVERIFY	18	3	0	0	0
148	USCHKIDLS	CSVERIFY	21	3	0	0	0
149	USCHKRLIS	CSVERIFY	19	4	0	0	0
150	USDERIFY	CSVERIFY	648	328	9	6	.014

	U6_ERRORS/LOC
121	0
122	0
123	0
124	0
125	.018
126	0
127	.005
128	0
129	0
130	0
131	0
132	0
133	0
134	0
135	.071
136	.011
137	.005
138	.004
139	.125
140	.024
141	.005
142	0
143	0
144	.020
145	0
146	0
147	0
148	0
149	0
150	.009

	UNIT	CSC	LOC	SH	ALL ERRORS	D6_ERRORS	ALL ERRORS/LOC
151	USCHKREAL	CSVERIFY	13	2	0	0	0
152	USCHKRSGP	CSVERIFY	21	4	1	0	.048
153	USCHKYORN	CSVERIFY	13	2	0	0	0
154	USWRITE	CSWRITE	389	43	9	4	.023

	U6_ERndRS/LOC
151	0
152	0
153	0
154	.010

APPENDIX D

CSC-level Analysis Data File

	CSC	LOC	SH	ALL ERRORS	N_UNITS	U6_ERRORS	LOC/UNIT	ALL ERRORS/LOC
1	CSINASAB	252	46	3	1	0	252.000	.012
2	CSINEDENT	3584	538	55	12	24	298.667	.015
3	CSINFANSEN	328	61	5	1	0	328.000	.015
4	CSINFIL	989	192	23	3	11	329.667	.023
5	CSINFLAGS	141	31	7	1	3	141.000	.050
6	CSINITIUT	5602	903	147	49	53	114.327	.026
7	CSINMODE	422	71	17	4	3	105.500	.640
8	CSINOWNOP	265	48	0	1	0	265.000	0
9	CSINPALA	250	40	0	1	0	250.000	0
10	CSINPSAT	836	98	9	2	2	418.000	.011
11	CSINF..SBS	1225	142	17	5	11	245.000	.014
12	CSINSENS	379	65	6	1	0	379.000	.016
13	CSOUTFIL	832	92	25	2	11	416.000	.030
14	CSHTDOWN	43	9	3	1	0	43.000	.070
	CSUNKDOWN	1316	243	•	•	•	•	•
16	CSUTILITIES	4329	572	90	43	30	100.674	.021
17	CSVERIFY	744	352	10	9	6	87.111	.013
18	CSWRITE	389	43	9	1	4	389.000	.023

	SH/N_UNITS	U6_ERRORS/LOC
1	46.000	0
2	44.87	.007
3	61.00	0
4	64.001	.011
5	31.004	.021
6	18.42	.069
7	17.756	.007
8	48.000	0
9	40.000	0
10	49.000	.002
11	28.400	.009
12	65.000	0
13	46.000	.013
14	9.000	0
	•	•
16	13.302	.007
17	39.111	.008
18	43.000	.010